xroms

Release 0.6.0

Rob Hetland, Kristen Thyng, Veronica Ruiz Xomchuk

Mar 04, 2024

EXAMPLES AND DEMOS

CHAPTER

ONE

INSTALLATION

To install from conda-forge:

```
>>> conda install -c conda-forge xroms
```

To install from PyPI:

```
>>> pip install xroms
```

1.1 How to load data

You should read in your model output of choice using `xarray`; more information on input/output with `xarray` can be found here.

Note: There are a few functions to read in model output with `xroms` but they are scheduled to be removed in future versions of `xroms`.

1.1.1 Some specific notes

Chunks

Chunks are used to break up model output into smaller units for use with `dask`. Inputting chunks into a call to open a dataset requires the use of `dask`. This can be formalized more by setting up a `dask` cluster. The best sizing of chunks is not clear *a priori* and requires some testing.

`open_mfdataset()`

Some useful keyword argument selections for when using `xr.open_mfdataset()` are suggested here:

```
{'compat': 'override', 'combine': 'by_coords',
     'data_vars': 'minimal', 'coords': 'minimal', 'parallel': True}
```

For example,

```
xr.open_mfdataset(url, compat='override', combine='by_coords',
     data_vars='minimal', coords='minimal', parallel=True}
```

`open_zarr()`

Some useful keyword argument selections are for reading in files with `xr.open_zarr()` are:

```
{'consolidated': True, 'drop_variables': 'dstart'}
```

and for concatenating the files together:

```
{'dim': 'ocean_time', 'data_vars': 'minimal', 'coords': 'minimal'}
```

1.1.2 Suggested Workflow for `xroms`:

1. Read in model output using the appropriate `xarray` function.
2. Supplement your Dataset and calculate an `xgcm` grid object with:

   ds, xgrid = xroms.roms_dataset(ds)

The function adds z coordinates and other useful metrics to the `Dataset`, including the z coordinates on each horizontal grid (e.g., `z_rho_u`), and the z coordinates relative to mean sea level (e.g., `z_rho0`). It also sets up an xgcm grid object for the Dataset, which is stored necessary for many `xroms` functions, and can be stored and accessible in the accessor (`ds.xroms.xgrid`).

There are optional flags for `xroms.roms_dataset()` for what all metrics to lazily calculate since it can be time-consuming despite being lazily loaded and calculated; see the *API docs* for details.

Alternatively, `roms_dataset()` will be run automatically the first time you use the Dataset accessor and `xgrid` will be stored in the object. Note that the default input flags to `roms_dataset()` are used in the case and if you want to have more control over that, you can use the following to override the xgrid stored

```
ds, xgrid = xroms.roms_dataset(ds, [other flags you want to use])
ds.xroms.set_grid(xgrid)
```

1.1.3 Demo workflow using example dataset

```
import xarray as xr
import xroms

ds = xroms.datasets.fetch_ROMS_example_full_grid()
ds, xgrid = xroms.roms_dataset(ds, include_cell_volume=True)
ds.xroms.set_grid(xgrid)
```

1.1.4 Save output

After model output has been read in with `xarray`, it can be used for calculations and/or subsetted, then easily saved back out to a file (in this case saving out only the first time step):

```
ds.isel(ocean_time=0).to_netcdf('filename.nc')
```

```
import xarray as xr
import xroms
import pandas as pd
```

(continues on next page)

(continued from previous page)

```
import numpy as np
import matplotlib.pyplot as plt
import cartopy
```

1.2 How to select data

The *input/output* notebook demonstrates how to load in data, but now how to select and slice it apart? Much of this is accomplished with the `sel` and `isel` methods in `xarray`, which are demonstrated in detail in this notebook.

Use `sel` to select/slice a Dataset or DataArray by dimension values; the best example of this for ROMS output is selecting certain time using a string representation of a datetime.

```
ds.salt.sel(ocean_time='2010-1-1 12:00')
ds.salt.sel(ocean_time=slice('2010-1-1', '2010-2-1'))
```

Use `isel` to subdivide a Dataset or DataArray by dimension indices:

```
ds.salt.isel(eta_rho=20, xi_rho=100)
ds.salt.isel(eta_rho=slice(20,100,10), xi_rho=slice(None,None,5))
```

1.2.1 cf-xarray

`xroms` includes the `cf-xarray` accessor, which allows you to use xarray `sel` and `isel` commands for a DataArray without needing to input the exact grid – just the axes.

With `xarray` alone:

```
ds.salt.isel(xi_rho=20, eta_rho=10, s_rho=20, ocean_time=10)
```

With `cf-xarray` accessor:

```
ds.salt.cf.isel(X=20, Y=10, Z=20, T=10)
```

and get the same thing back. Same for `sel`. The T, Z, Y, X names can be mixed and matched with the actual dimension names. Some of the attribute wrangling in `xroms` is dedicated to making sure that `cf-xarray` can always identify dimensions and coordinates for DataArrays.

You can always check what `cf-xarray` understands about a Dataset or DataArray with

```
ds.salt.cf.describe()
```

Load in data

More information at in *input/output page*

```
ds = xroms.datasets.fetch_ROMS_example_full_grid()
ds, xgrid = xroms.roms_dataset(ds, include_cell_volume=True)
ds.xroms.set_grid(xgrid)
ds
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
```

(continues on next page)

(continued from previous page)

```
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
```

(continues on next page)

(continued from previous page)

```
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

(continues on next page)

(continued from previous page)

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
```

(continues on next page)

(continued from previous page)

```
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
<xarray.Dataset> Size: 734MB
Dimensions:     (eta_rho: 191, xi_rho: 300, s_rho: 30, s_w: 31, ocean_time: 2,
                 xi_u: 299, eta_v: 190)
Coordinates: (12/21)
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_rho       (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * s_w         (s_w) float64 248B -1.0 -0.9667 -0.9333 ... -0.03333 0.0
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
    lon_u       (eta_rho, xi_u) float64 457kB dask.array<chunksize=(191, 299), meta=np.
↪ndarray>
    ...          ...
    z_w_v       (ocean_time, s_w, eta_v, xi_rho) float64 28MB dask.array<chunksize=(2,␣
↪31, 190, 300), meta=np.ndarray>
    z_w_psi     (ocean_time, s_w, eta_v, xi_u) float64 28MB dask.array<chunksize=(2, 31,␣
↪190, 299), meta=np.ndarray>
    z_rho       (ocean_time, s_rho, eta_rho, xi_rho) float64 28MB dask.array
↪<chunksize=(2, 30, 191, 300), meta=np.ndarray>
    z_rho_u     (ocean_time, s_rho, eta_rho, xi_u) float64 27MB dask.array<chunksize=(2,␣
↪30, 191, 299), meta=np.ndarray>
    z_rho_v     (ocean_time, s_rho, eta_v, xi_rho) float64 27MB dask.array<chunksize=(2,␣
↪30, 190, 300), meta=np.ndarray>
    z_rho_psi   (ocean_time, s_rho, eta_v, xi_u) float64 27MB dask.array<chunksize=(2,␣
↪30, 190, 299), meta=np.ndarray>
Data variables: (12/45)
    angle       (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    hc          float64 8B ...
    Cs_r        (s_rho) float64 240B dask.array<chunksize=(30,), meta=np.ndarray>
    zeta        (ocean_time, eta_rho, xi_rho) float32 458kB dask.array<chunksize=(2, 191,
↪ 300), meta=np.ndarray>
    h           (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    Cs_w        (s_w) float64 248B dask.array<chunksize=(31,), meta=np.ndarray>
    ...          ...
    dV_w_u      (ocean_time, s_w, eta_rho, xi_u) float64 28MB dask.array<chunksize=(2,␣
↪31, 191, 299), meta=np.ndarray>
    dV_v        (ocean_time, s_rho, eta_v, xi_rho) float64 27MB dask.array<chunksize=(2,␣
↪30, 190, 300), meta=np.ndarray>
    dV_w_v      (ocean_time, s_w, eta_v, xi_rho) float64 28MB dask.array<chunksize=(2,␣
↪31, 190, 300), meta=np.ndarray>
    dV_psi      (ocean_time, s_rho, eta_v, xi_u) float64 27MB dask.array<chunksize=(2,␣
↪30, 190, 299), meta=np.ndarray>
```

(continues on next page)

(continued from previous page)

```
    dV_w_psi    (ocean_time, s_w, eta_v, xi_u) float64 28MB dask.array<chunksize=(2, 31,␣
↪190, 299), meta=np.ndarray>
    rho0        int64 8B 1025
Attributes: (12/29)
    file:              ocean_his_0150.nc
    format:            netCDF-3 classic file
    Conventions:       CF-1.4
    type:              ROMS/TOMS history file
    title:             Texas and Louisiana Shelf case (Nesting)
    rst_file:          ocean_rst.nc
    ...                ...
    compiler_command:  /g/software/openmpi/1.4.3/intel/bin/mpif90
    compiler_flags:    -heap-arrays -fp-model precise -assume 2underscores -c...
    tiling:            016x032
    history:           ROMS/TOMS, Version 3.4, Sunday - December 4, 2011 -  7...
    ana_file:          /scratch/zhangxq/projects/txla_nesting6/Functionals/an...
    CPP_options:       TXLA, ANA_BSFLUX, ANA_BTFLUX, ASSUMED_SHAPE, BULK_FLUX...
```

```
ds.salt.cf.describe()
```

```
Coordinates:
            CF Axes: * X: ['xi_rho']
                     * Y: ['eta_rho']
                     * Z: ['s_rho']
                     * T: ['ocean_time']

      CF Coordinates:   longitude: ['lon_rho']
                        latitude: ['lat_rho']
                        vertical: ['z_rho']
                      * time: ['ocean_time']

       Cell Measures:   area, volume: n/a

      Standard Names:   depth: ['z_rho']
                        latitude: ['lat_rho']
                        longitude: ['lon_rho']
                      * ocean_s_coordinate_g1: ['s_rho']
                      * time: ['ocean_time']

              Bounds:   n/a

       Grid Mappings:   n/a
```

```
/tmp/ipykernel_2936/990108105.py:1: DeprecationWarning: 'obj.cf.describe()' will be␣
↪removed in a future version. Use instead 'repr(obj.cf)' or 'obj.cf' in a Jupyter␣
↪environment.
  ds.salt.cf.describe()
```

1.2.2 Select

Surface layer slice

The surface in ROMS is given by the last index in the vertical dimension. The easiest way to access this is by indexing into `s_rho`. While normally it is better to access coordinates through keywords to be human-readable, it's not easy to tell what value of `s_rho` gives the surface. In this instance, it's easier to just go by index.

```
ds.salt.isel(s_rho=-1)

ds.salt.cf.isel(Z=-1)  # with cf-xarray
```

You can also grab the Z level that is "nearest" to 0, the surface, which will give the same vertical level as the other options:

```
ds.salt.cf.sel(Z=0, method="nearest")
```

```
ds.salt.cf.sel(Z=0, method="nearest")
```

```
<xarray.DataArray 'salt' (ocean_time: 2, eta_rho: 191, xi_rho: 300)> Size: 458kB
dask.array<getitem, shape=(2, 191, 300), dtype=float32, chunksize=(2, 191, 300),␣
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    s_rho       float64 8B -0.01667
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    z_rho       (ocean_time, eta_rho, xi_rho) float64 917kB dask.array<chunksize=(2, 191,
↪ 300), meta=np.ndarray>
Attributes:
    long_name:  salinity
    time:       ocean_time
    field:      salinity, scalar, series
```

x/y index slice

For a curvilinear ROMS grid, selecting by the dimensions `xi_rho` or `eta_rho` (or for whichever is the relevant grid) is not very meaningful because they are given by index. Thus the following is possible to get a slice along the index, but it cannot be used to find a slice based on the lon/lat values. For the eta and xi grids, `sel` is equivalent to `isel`.

```
ds.temp.sel(xi_rho=20)

ds.temp.cf.sel(X=20);  # same with cf-xarray accessor
```

```
ds.temp.cf.sel(X=20)
```

```
<xarray.DataArray 'temp' (ocean_time: 2, s_rho: 30, eta_rho: 191)> Size: 46kB
dask.array<getitem, shape=(2, 30, 191), dtype=float32, chunksize=(2, 30, 191),␣
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho) float64 2kB dask.array<chunksize=(191,), meta=np.ndarray>
    lat_rho     (eta_rho) float64 2kB dask.array<chunksize=(191,), meta=np.ndarray>
  * s_rho       (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
    xi_rho      int64 8B 20
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    z_rho       (ocean_time, s_rho, eta_rho) float64 92kB dask.array<chunksize=(2, 30,␣
↪191), meta=np.ndarray>
Attributes:
    long_name:  potential temperature
    units:      Celsius
    time:       ocean_time
    field:      temperature, scalar, series
```

Single time

Find the forecast model output available that is closest to now. Note that the `method` keyword argument is not necessary if the desired date/time is exactly a model output time. You can daisy-chain together different `sel` and `isel` calls.

```
date = "2009-11-19T13:00"

ds.salt.isel(s_rho=-1).sel(ocean_time=date, method='nearest')

ds.salt.cf.isel(Z=-1).cf.sel(T=date, method='nearest')  # with cf-xarray
```

```
date = "2009-11-19T13:00"
ds.salt.cf.sel(Z=0, T=date, method='nearest')
```

```
<xarray.DataArray 'salt' (eta_rho: 191, xi_rho: 300)> Size: 229kB
dask.array<getitem, shape=(191, 300), dtype=float32, chunksize=(191, 300),␣
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    s_rho       float64 8B -0.01667
    ocean_time  datetime64[ns] 8B 2009-11-19T12:00:00
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    z_rho       (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
Attributes:
    long_name:  salinity
    time:       ocean_time
    field:      salinity, scalar, series
```

Range of time

```
time_range = slice(date, pd.Timestamp(date)+pd.Timedelta('3 hours'))

ds.salt.sel(ocean_time=time_range)

ds.salt.cf.sel(T=time_range)  # cf-xarray
```

```
time_range = slice(date, pd.Timestamp(date)+pd.Timedelta('3 hours'))
ds.salt.cf.sel(T=time_range)  # cf-xarray
```

```
<xarray.DataArray 'salt' (ocean_time: 1, s_rho: 30, eta_rho: 191, xi_rho: 300)> Size: 7MB
dask.array<getitem, shape=(1, 30, 191, 300), dtype=float32, chunksize=(1, 30, 191, 300),␣
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_rho       (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * ocean_time  (ocean_time) datetime64[ns] 8B 2009-11-19T16:00:00
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    z_rho       (ocean_time, s_rho, eta_rho, xi_rho) float64 14MB dask.array
↪<chunksize=(1, 30, 191, 300), meta=np.ndarray>
Attributes:
    long_name:  salinity
    time:       ocean_time
    field:      salinity, scalar, series
```

Select region

Select a boxed region by min/max lon and lat values.

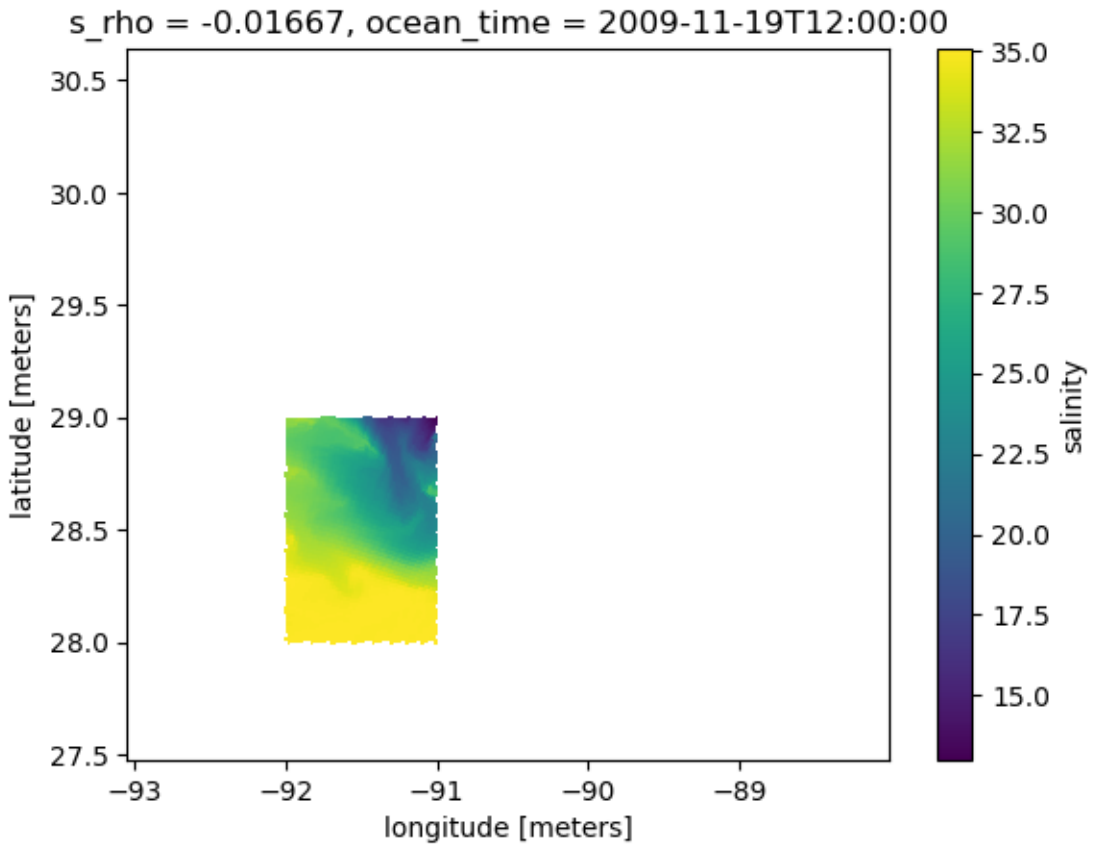
```
# want model output only within the box defined by these lat/lon values
lon = np.array([-92, -91])
lat = np.array([28, 29])
```

```
# this condition defines the region of interest
box = ((lon[0] < ds["salt"].cf["longitude"]) & (ds["salt"].cf["longitude"] < lon[1])
       & (lat[0] < ds["salt"].cf["latitude"]) & (ds["salt"].cf["latitude"] < lat[1])).
↪compute()
```

Plot the model output in the box at the surface

```
dss = ds.where(box).salt.cf.isel(Z=-1, T=0)
dss.cf.plot(x='longitude', y='latitude')
```

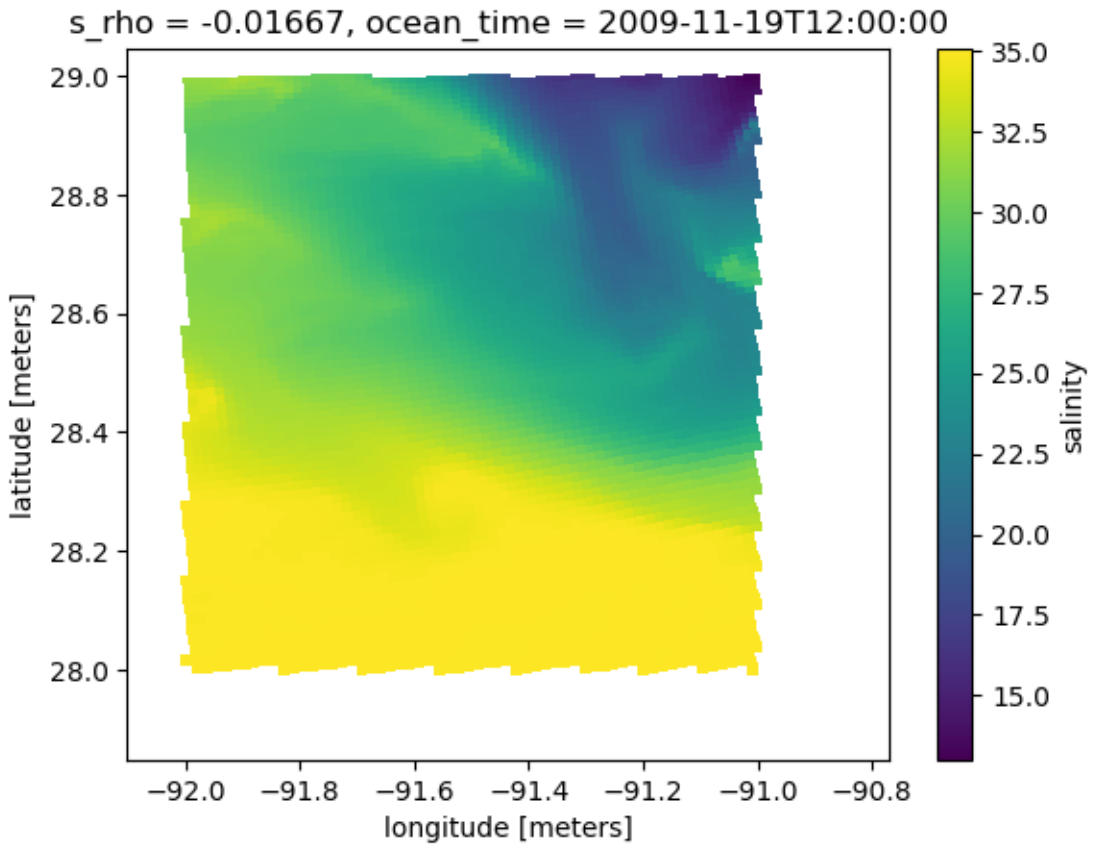
```
<matplotlib.collections.QuadMesh at 0x7f4b442dce80>
```



If you don't need the rest of the model output, you can drop it by using `drop=True` in the `where` call.

```
dss = ds.where(box, drop=True).salt.cf.isel(Z=-1, T=0)
dss.cf.plot(x='longitude', y='latitude')
```

```
<matplotlib.collections.QuadMesh at 0x7f4b44227670>
```



Can calculate a metric within the box:

```
dss.mean().values
```

```
array(28.308672, dtype=float32)
```

Subset model output

Subset Dataset of model output such that subsetted domain is as if the simulation was run on that size grid. That is, the rho grid is 1 larger than the psi grid in each of xi and eta.

```
ds.xroms.subset(X=slice(20,40), Y=slice(50,100))  # with accessor

xroms.subset(ds, X=slice(20,40), Y=slice(50,100))
```

```
ds.xroms.subset(X=slice(20,40), Y=slice(50,100))  # with accessor
```

```
<xarray.Dataset> Size: 12MB
Dimensions:     (eta_rho: 50, xi_rho: 20, s_rho: 30, s_w: 31, ocean_time: 2,
                 xi_u: 19, eta_v: 49)
Coordinates: (12/21)
    lon_rho     (eta_rho, xi_rho) float64 8kB dask.array<chunksize=(50, 20), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 8kB dask.array<chunksize=(50, 20), meta=np.
```

(continues on next page)

(continued from previous page)

```
↪ndarray>
  * s_rho       (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * s_w         (s_w) float64 248B -1.0 -0.9667 -0.9333 ... -0.03333 0.0
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
    lon_u       (eta_rho, xi_u) float64 8kB dask.array<chunksize=(50, 19), meta=np.
↪ndarray>
    ...          ...
    z_w_v       (ocean_time, s_w, eta_v, xi_rho) float64 486kB dask.array<chunksize=(2,␣
↪31, 49, 20), meta=np.ndarray>
    z_w_psi     (ocean_time, s_w, eta_v, xi_u) float64 462kB dask.array<chunksize=(2, 31,
↪ 49, 19), meta=np.ndarray>
    z_rho       (ocean_time, s_rho, eta_rho, xi_rho) float64 480kB dask.array
↪<chunksize=(2, 30, 50, 20), meta=np.ndarray>
    z_rho_u     (ocean_time, s_rho, eta_rho, xi_u) float64 456kB dask.array<chunksize=(2,
↪ 30, 50, 19), meta=np.ndarray>
    z_rho_v     (ocean_time, s_rho, eta_v, xi_rho) float64 470kB dask.array<chunksize=(2,
↪ 30, 49, 20), meta=np.ndarray>
    z_rho_psi   (ocean_time, s_rho, eta_v, xi_u) float64 447kB dask.array<chunksize=(2,␣
↪30, 49, 19), meta=np.ndarray>
Data variables: (12/45)
    angle       (eta_rho, xi_rho) float64 8kB dask.array<chunksize=(50, 20), meta=np.
↪ndarray>
    hc          float64 8B ...
    Cs_r        (s_rho) float64 240B dask.array<chunksize=(30,), meta=np.ndarray>
    zeta        (ocean_time, eta_rho, xi_rho) float32 8kB dask.array<chunksize=(2, 50,␣
↪20), meta=np.ndarray>
    h           (eta_rho, xi_rho) float64 8kB dask.array<chunksize=(50, 20), meta=np.
↪ndarray>
    Cs_w        (s_w) float64 248B dask.array<chunksize=(31,), meta=np.ndarray>
    ...          ...
    dV_w_u      (ocean_time, s_w, eta_rho, xi_u) float64 471kB dask.array<chunksize=(2,␣
↪31, 50, 19), meta=np.ndarray>
    dV_v        (ocean_time, s_rho, eta_v, xi_rho) float64 470kB dask.array<chunksize=(2,
↪ 30, 49, 20), meta=np.ndarray>
    dV_w_v      (ocean_time, s_w, eta_v, xi_rho) float64 486kB dask.array<chunksize=(2,␣
↪31, 49, 20), meta=np.ndarray>
    dV_psi      (ocean_time, s_rho, eta_v, xi_u) float64 447kB dask.array<chunksize=(2,␣
↪30, 49, 19), meta=np.ndarray>
    dV_w_psi    (ocean_time, s_w, eta_v, xi_u) float64 462kB dask.array<chunksize=(2, 31,
↪ 49, 19), meta=np.ndarray>
    rho0        int64 8B 1025
Attributes: (12/29)
    file:              ocean_his_0150.nc
    format:            netCDF-3 classic file
    Conventions:       CF-1.4
    type:              ROMS/TOMS history file
    title:             Texas and Louisiana Shelf case (Nesting)
    rst_file:          ocean_rst.nc
    ...                ...
    compiler_command:  /g/software/openmpi/1.4.3/intel/bin/mpif90
    compiler_flags:    -heap-arrays -fp-model precise -assume 2underscores -c...
    tiling:            016x032
```

(continues on next page)

(continued from previous page)

```
    history:           ROMS/TOMS, Version 3.4, Sunday - December 4, 2011 -  7...
    ana_file:          /scratch/zhangxq/projects/txla_nesting6/Functionals/an...
    CPP_options:       TXLA, ANA_BSFLUX, ANA_BTFLUX, ASSUMED_SHAPE, BULK_FLUX...
```

Find nearest in lon/lat

This matters for a curvilinear grid.

Can't use `sel` because it will only search in one dimension for the nearest value and the dimensions are indices which are not necessarily geographic distance. Instead need to use a search for distance and use that for the `where` condition from the previous example. This functionality has been wrapped into `xroms.sel2d` (and its partner function `xroms.argsel2d`).

```
lon0, lat0 = -91, 28
saltsel = ds.salt.xroms.sel2d(lon0, lat0)
```

Or, if you instead want the indices of the nearest grid node returned, you can call `argsel2d`:

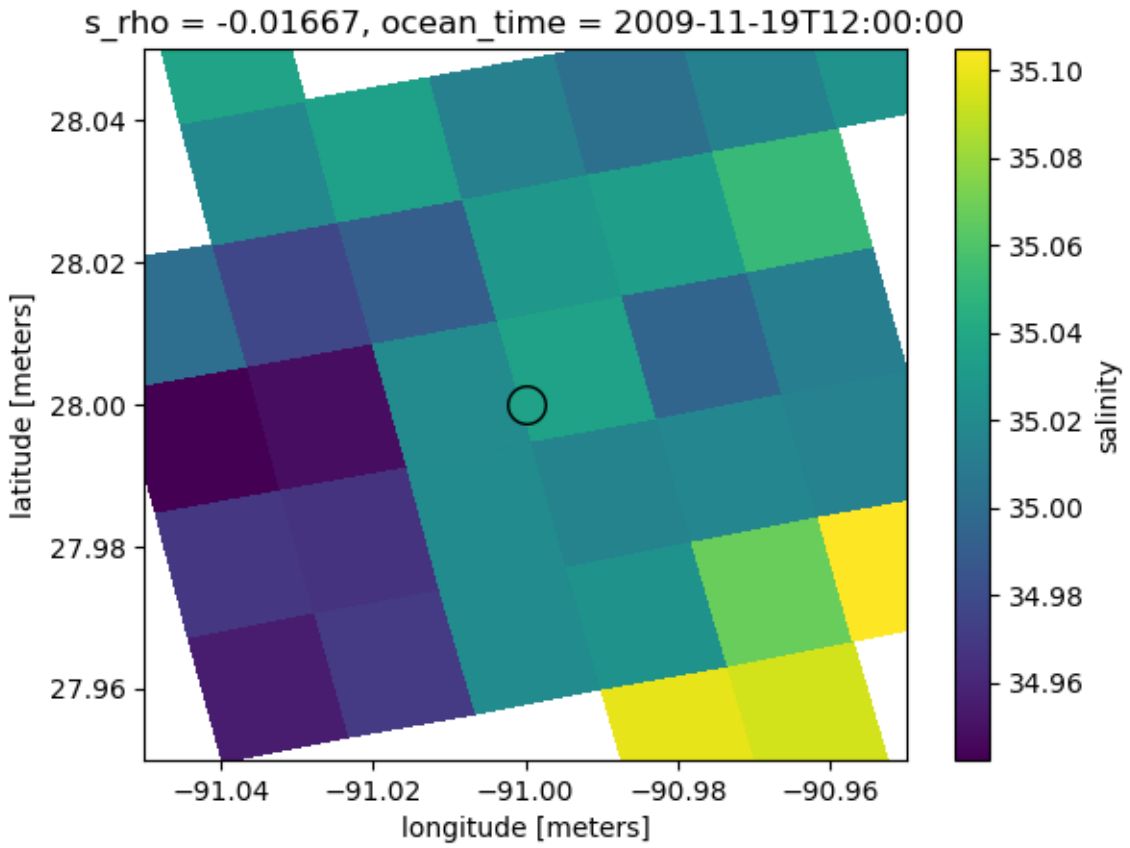
```
ds.salt.xroms.argsel2d(lon0, lat0)
```

```
(16, 110)
```

Check this function, just to be sure:

```
dl = 0.05
box = (ds.lon_rho>lon0-dl) & (ds.lon_rho<lon0+dl) & (ds.lat_rho>lat0-dl) & (ds.lat_rho
↪<lat0+dl)
dss = ds.where(box).salt.cf.isel(T=0, Z=-1)

vmin = dss.min().values
vmax = dss.max().values

dss.plot(x='lon_rho', y='lat_rho')
plt.scatter(lon0, lat0, c=saltsel.cf.isel(Z=-1, T=0), s=200, edgecolor='k', vmin=vmin,␣
↪vmax=vmax)
plt.xlim(lon0-dl,lon0+dl)
plt.ylim(lat0-dl, lat0+dl)
```

```
(27.95, 28.05)
```




```
import xarray as xr
import xgcm
import numpy as np
import xroms
import matplotlib.pyplot as plt
```

```
Matplotlib is building the font cache; this may take a moment.
```

1.3 How to calculate with `xarray` and `xroms`

Here we demonstrate a number of calculations built into `xroms`, through accessors to `DataArrays` and `Datasets`.

1.3.1 `xarray` Datasets

Use an `xarray` accessor in `xroms` to easily perform calculations with syntax

```
ds.xroms.[method]
```

Importantly, the `xroms` accessor to a `Dataset` is initialized with an `xgcm` grid object (or you can input a previously-calculated grid object), stored at `ds.xroms.xgrid`, which is used to perform the basic grid calculations. More on this under "How to set up grid" below.

The built-in native calculations are properties of the `xroms` accessor and are not functions.

The accessor functions can take in the horizontal then vertical grid label you want the calculation to be on as options:

```
ds.xroms.ddz('u', hcoord='rho', scoord='s_rho')  # to make sure result is on rho␣
↪horizontal grid and s_rho vertical grid, a function
```

or

```
ds.xroms.dudz  # return on native grid it is calculated on, a property
```

Other inputs are available for functions when the calculation involves a derivative and there is a choice for how to treat the boundary (`hboundary` and `hfill_value` for horizontal calculations and `sboundary` and `sfill_value` for vertical calculations). More information on those inputs can be found in the docs for `xgcm` such as under found under:

```
ds.xroms.xgrid.interp?
```

1.3.2 `xarray` DataArrays

A few of the more basic methods in `xroms` are available to `DataArrays` too. `xroms` methods for `DataArrays` require the grid object to be input:

```
ds.temp.xroms.to_grid(xgrid, hcoord='psi', scoord='s_w')
```

1.3.3 Attributes

`xroms` provides attributes as metadata to track calculations, provide context, and to be used as indicators for plots.

The option to always keep attributes in `xarray` is turned on in the call to `xroms`.

1.3.4 `cf-xarray`

Some functionality is added by using the package `cf-xarray`. Necessary attributes are added to datasets when the following call is run:

```
ds, xgrid = xroms.roms_dataset(ds)
```

For example, when all CF Convention attributes are available in the Dataset, you can refer to dimensions and coordinates generically, regardless of the actual variable names.

- For dimensions:
  - ds.cf[“T”], ds.cf[“Z”], ds.cf[“Y”], ds.cf[“X”]
- For coordinates:
  - ds.cf[“time], ds.cf[“vertical”], ds.cf[“latitude”], ds.cf[“longitude”]

Load in data

More information on input/output in *input/output page*. For model output available at , you can find your dataset with and chunks according to the dataset itself (though if none are known by the dataset this will use no chunks):

```
ds = xr.open_dataset(url, chunks={})
```

Also, an example ROMS dataset is available with `xroms` that we will read in for this tutorial.

```
ds = xroms.datasets.fetch_ROMS_example_full_grid()
ds
```

```
Downloading file 'ROMS_example_full_grid.nc' from 'https://github.com/xoceanmodel/xroms/
↪raw/main/xroms/data/ROMS_example_full_grid.nc' to '/home/docs/.cache/xroms'.
```

```
<xarray.Dataset> Size: 62MB
Dimensions:     (eta_rho: 191, xi_rho: 300, s_rho: 30, s_w: 31, ocean_time: 2,
                 eta_u: 191, xi_u: 299, eta_v: 190, xi_v: 300)
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_rho       (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * s_w         (s_w) float64 248B -1.0 -0.9667 -0.9333 ... -0.03333 0.0
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
    lon_u       (eta_u, xi_u) float64 457kB dask.array<chunksize=(191, 299), meta=np.
↪ndarray>
    lat_u       (eta_u, xi_u) float64 457kB dask.array<chunksize=(191, 299), meta=np.
↪ndarray>
    lon_v       (eta_v, xi_v) float64 456kB dask.array<chunksize=(190, 300), meta=np.
↪ndarray>
    lat_v       (eta_v, xi_v) float64 456kB dask.array<chunksize=(190, 300), meta=np.
↪ndarray>
Dimensions without coordinates: eta_rho, xi_rho, eta_u, xi_u, eta_v, xi_v
Data variables: (12/17)
    angle       (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    hc          float64 8B ...
    Cs_r        (s_rho) float64 240B dask.array<chunksize=(30,), meta=np.ndarray>
    zeta        (ocean_time, eta_rho, xi_rho) float32 458kB dask.array<chunksize=(2, 191,
↪ 300), meta=np.ndarray>
    h           (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    Cs_w        (s_w) float64 248B dask.array<chunksize=(31,), meta=np.ndarray>
    ...          ...
    temp        (ocean_time, s_rho, eta_rho, xi_rho) float32 14MB dask.array
↪<chunksize=(2, 30, 191, 300), meta=np.ndarray>
    salt        (ocean_time, s_rho, eta_rho, xi_rho) float32 14MB dask.array
↪<chunksize=(2, 30, 191, 300), meta=np.ndarray>
    Vtransform  int32 4B ...
    pm          (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
```

(continues on next page)

(continued from previous page)

```
    pn          (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    f           (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
Attributes: (12/29)
    file:               ocean_his_0150.nc
    format:             netCDF-3 classic file
    Conventions:        CF-1.4
    type:               ROMS/TOMS history file
    title:              Texas and Louisiana Shelf case (Nesting)
    rst_file:           ocean_rst.nc
    ...                 ...
    compiler_command:   /g/software/openmpi/1.4.3/intel/bin/mpif90
    compiler_flags:     -heap-arrays -fp-model precise -assume 2underscores -c...
    tiling:             016x032
    history:            ROMS/TOMS, Version 3.4, Sunday - December 4, 2011 -  7...
    ana_file:           /scratch/zhangxq/projects/txla_nesting6/Functionals/an...
    CPP_options:        TXLA, ANA_BSFLUX, ANA_BTFLUX, ASSUMED_SHAPE, BULK_FLUX...
```

```
ds, xgrid = xroms.roms_dataset(ds, include_cell_volume=True)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

(continues on next page)

(continued from previous page)

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
```

(continues on next page)

(continued from previous page)

```
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```python
# add grid to xrom accessor explicitly
ds.xroms.set_grid(xgrid)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
→packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
→be changed to return a set of dimension names in future, in order to be more␣
→consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
→please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
→packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
→be changed to return a set of dimension names in future, in order to be more␣
→consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
→please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
→packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
→be changed to return a set of dimension names in future, in order to be more␣
→consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
→please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
→packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
→be changed to return a set of dimension names in future, in order to be more␣
→consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
→please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
→packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
→be changed to return a set of dimension names in future, in order to be more␣
→consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
→please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
→packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
→be changed to return a set of dimension names in future, in order to be more␣
→consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
→please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
→packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
→be changed to return a set of dimension names in future, in order to be more␣
→consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
→please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
→packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
→be changed to return a set of dimension names in future, in order to be more␣
→consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
→please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
```

(continues on next page)

(continued from previous page)

```
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
```

(continues on next page)

(continued from previous page)

```
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
ds.xroms.xgrid
```

```
<xgcm.Grid>
X Axis (not periodic, boundary=None):
  * center   xi_rho --> inner
  * inner    xi_u --> center
Y Axis (not periodic, boundary=None):
  * center   eta_rho --> inner
  * inner    eta_v --> center
Z Axis (not periodic, boundary=None):
  * center   s_rho --> outer
  * outer    s_w --> center
```

1.3.5 xgcm grid and extra ROMS coordinates

How to set up grid

The package xcgm has many nice grid functions for ROMS users, however, a bit of set up is required to connect from ROMS to the xgcm standardds. This grid set up does that.

The grid object contains metrics (X, Y, Z) with distances for each grid ('dx', 'dx_u', 'dx_v', 'dx_psi', and 'dz', 'dz_u', 'dz_v', 'dz_w', 'dz_w_u', 'dz_w_v', 'dz_psi', 'dz_w_psi'), and all of these as grid coordinates too.

After setting up your Dataset, you should add coordinates and other information to the dataset and set up an xgcm grid object with:

```
ds, xgrid = xroms.roms_dataset(ds, include_cell_volume=True)
```

If you want to use the xroms accessor, add the grid object explicitly with:

```
ds.xroms.set_grid(xgrid)
```

If you don't do this step, the first time the grid object is required it will be set up, though you can't choose which input flags to use in that case.

The xgcm grid object is then available at

```
ds.xroms.xgrid
```

Grid lengths

Distances between grid nodes on every ROMS grid can be calculated and set up in the xgcm grid object — some by default and some have to be requested by the user with optional flags.

- Horizontal grids:
- inverse distances between nodes are given in an analogous way to distance (*i.e.*, ds.pm and ds.pn_psi)
- distances between nodes are given in meters by dx's and dy's stored in ds, such as: ds.dx for the rho grid and ds.dy_psi for the psi grid, calculated from inverse distances
- Vertical grids:
- There are lazily-evaluated z-coordinates for both rho and w vertical grids for each horizontal grid.
- There are also arrays of z distances between nodes, called dz's, available for each combination of grids. For example, there is ds.dz_u for z distances on the u horizontal and rho vertical grid, and there is ds.dz_w_v for z distances on the v horizontal and w vertical grid. These are [ocean_time x s_* x eta_* x xi_*] arrays.
- Arrays of z distances relative to a sea level of 0 are also available. They have analogous names to the previous entries but with "0" on the end. They are computationally faster to use because they do not vary in time. They are also less accurate for this reason but it depends on your use as to how much that matters.

Grid areas

- Horizontal
  - rho grid `ds.dA`, psi grid `ds.dA_psi`, u grid `ds.dA_u`, v grid `ds.dA_v`
- Vertical
  - These aren't built in but can easily be calculated. For example, for cell areas in the x direction on the rho horizontal and rho vertical grids: `ds.dx * ds.dz`.

Grid volumes

Time varying: All 8 combinations of 4 horizontal grids and 2 vertical grids are available if `include_cell_volume==True` in `roms_dataset()`, such as: `ds.dV` (rho horizontal, rho vertical), and `ds.dV_w_v` (w vertical, v horizontal).

A user can easily calculate the same but for time-constant dz's, for example as:

```
ds['dV_w'] = ds.dx * ds.dy * ds.dz_w0  # w vertical, rho horizontal, constant in time
```

You can calculate the full domain volume in time with:

```
ds.dV.sum(('s_rho', 'eta_rho', 'xi_rho'))
```

Or, using `cf-xarray` with:

```
ds.dV.cf.sum(('Z', 'Y', 'X'))
```

```
ds.dV.cf.sum(('Z', 'Y', 'X'))  # with cf-xarray accessor
```

```
<xarray.DataArray 'dV' (ocean_time: 2)> Size: 16B
dask.array<sum-aggregate, shape=(2,), dtype=float64, chunksize=(2,), chunktype=numpy.
↪ndarray>
Coordinates:
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
Attributes:
    long_name:  volume metric in XI and ETA and S on RHO/RHO grids
    units:      meter3
    field:      dV, scalar
```

1.3.6 Change grids

A ROMS user frequently needs to move between horizontal and vertical grids, so it is built into many of the function wrappers, but you can also do it as a separate function. It can also be done directly to `Datasets` with the `xroms` accessor. Here we change salinity from its default grids to be on the psi grid horizontally and the s_w grid vertically:

```
ds.xroms.to_grid('salt', 'psi', 's_w')
```

You can also use the `xroms` function directly instead of using the `xarray` accessor if you prefer to have more options. Here is the equivalent call to the accessor, using the same defaults:

```
xroms.to_grid(ds["salt"], xgrid,
              hcoord="psi", scoord="s_w",
              hboundary="extend", hfill_value=None,
              sboundary="extend", sfill_value=None)
```

```
ds.xroms.to_grid('salt', 'psi', 's_w')
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
<xarray.DataArray 'salt' (ocean_time: 2, s_w: 31, eta_v: 190, xi_u: 299)> Size: 14MB
dask.array<transpose, shape=(2, 31, 190, 299), dtype=float32, chunksize=(2, 31, 190,␣
↪299), chunktype=numpy.ndarray>
Coordinates:
  * s_w         (s_w) float64 248B -1.0 -0.9667 -0.9333 ... -0.03333 0.0
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_u        (xi_u) int64 2kB 0 1 2 3 4 5 6 7 ... 292 293 294 295 296 297 298
  * eta_v       (eta_v) int64 2kB 0 1 2 3 4 5 6 ... 183 184 185 186 187 188 189
    z_w_psi     (ocean_time, s_w, eta_v, xi_u) float64 28MB dask.array<chunksize=(2, 31,␣
↪190, 299), meta=np.ndarray>
Attributes:
    long_name:  salinity
    time:       ocean_time
    field:      salinity, scalar, series
    name:       salt
    units:      units
```

1.3.7 Dimension ordering convention

By convention, ROMS DataArrays should be in the order ['T', 'Z', 'Y', 'X'], for however many of these dimensions they contain. The following function does this for you:

```
xroms.order(ds.temp);  # function call

ds.temp.xroms.order();  # accessor
```

```
ds.temp.xroms.order()  # accessor
```

```
<xarray.DataArray 'temp' (ocean_time: 2, s_rho: 30, eta_rho: 191, xi_rho: 300)> Size:␣
↪14MB
dask.array<open_dataset-temp, shape=(2, 30, 191, 300), dtype=float32, chunksize=(2, 30,␣
↪191, 300), chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_rho       (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    z_rho       (ocean_time, s_rho, eta_rho, xi_rho) float64 28MB dask.array
↪<chunksize=(2, 30, 191, 300), meta=np.ndarray>
Attributes:
    long_name:  potential temperature
    units:      Celsius
    time:       ocean_time
    field:      temperature, scalar, series
```

1.3.8 Basic computations

These are all functions, not properties.

xarray

Many computations are built into `xarray` itself. Often it is possible to input the dimension over which to perform a computation by name, such as:

```
arr.sum(dim="xi_rho")
```

or

```
arr.sum(dim=("xi_rho","eta_rho"))
```

Note that many basic `xarray` calculations should be used with caution when using with ROMS output, since a ROMS grid can be stretched both horizontally and vertically. When using these functions, consider if your calculation should account for variable grid cell distances, areas, or volumes. Additionally, it is straight-forward to use basic grid functions from `xarray` on a ROMS time dimension (resampling, differentiation, interpolation, etc), however, be careful before using these functions on spatial dimensions for the same reasons as before.

```
ds.salt.mean(dim=("xi_rho","eta_rho"))

# same call but using cf-xarray
ds.salt.cf.mean(("Y","X"))
```

```
ds.salt.cf.mean(("Y","X"))
```

```
<xarray.DataArray 'salt' (ocean_time: 2, s_rho: 30)> Size: 240B
dask.array<mean_agg-aggregate, shape=(2, 30), dtype=float32, chunksize=(2, 30),␣
↪chunktype=numpy.ndarray>
Coordinates:
  * s_rho       (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
Attributes:
    long_name:  salinity
    time:       ocean_time
    field:      salinity, scalar, series
```

xroms grid-based metrics

Spatial metrics that account for the variable grid cell sizing in ROMS (both curvilinear horizontal and s vertical) are available by wrapping xgcm functions. These also have the additional benefit that the user can change grids and attributes are tracked. The available functions are:

- gridsum
- gridmean

Example usage:

```
xroms.gridsum(ds.temp, xgrid, dim)  # function call

ds['temp'].xroms.gridsum(xgrid, dim)  # accessor
```

where dimension names in the xgcm convention are 'Z', 'Y', or 'X'. dim can be a string, list, or tuple of combinations of these names for dimensions to average over.

sum

```
uint = ds.u.xroms.gridsum(xgrid, "Z")
```

```
uint.xroms.gridsum(xgrid, 'Y')
```

```
<xarray.DataArray (ocean_time: 2, xi_u: 299)> Size: 5kB
dask.array<sum-aggregate, shape=(2, 299), dtype=float64, chunksize=(2, 299),␣
↪chunktype=numpy.ndarray>
Coordinates:
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_u        (xi_u) int64 2kB 0 1 2 3 4 5 6 7 ... 292 293 294 295 296 297 298
Attributes:
```

(continues on next page)

(continued from previous page)

```
    long_name:  u-momentum component
    units:      meter second-1
    time:       ocean_time
    field:      u-velocity, scalar, series
```

mean

```
vint = ds.v.xroms.gridmean(xgrid, "Z")
vint.xroms.gridmean(xgrid, "Y")
```

```
<xarray.DataArray 'v' (ocean_time: 2, xi_rho: 300)> Size: 5kB
dask.array<truediv, shape=(2, 300), dtype=float64, chunksize=(2, 300), chunktype=numpy.
↪ndarray>
Coordinates:
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
Attributes:
    long_name:  v-momentum component
    units:      meter second-1
    time:       ocean_time
    field:      v-velocity, scalar, series
```

1.3.9 Derivatives

Vertical

Syntax is:

```
ds.xroms.ddz("salt")  # accessor to dataset

xroms.ddz(ds.salt, xgrid)  # No accessor
```

Other options:

```
ds.xroms.ddz('salt', hcoord='psi', scoord='s_rho', sboundary='extend', sfill_value=np.
↪nan);  # Dataset

xroms.ddz(ds.salt, xgrid, hcoord='psi', scoord='s_rho', sboundary='extend', sfill_
↪value=np.nan);  # No accessor
```

```
ds.xroms.ddz('salt')  # Dataset
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
<xarray.DataArray 'dsaltdz' (ocean_time: 2, s_w: 31, eta_rho: 191, xi_rho: 300)> Size:␣
↪28MB
dask.array<truediv, shape=(2, 31, 191, 300), dtype=float64, chunksize=(2, 31, 191, 300),␣
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_w         (s_w) float64 248B -1.0 -0.9667 -0.9333 ... -0.03333 0.0
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    z_w         (ocean_time, s_w, eta_rho, xi_rho) float64 28MB dask.array<chunksize=(2,␣
↪31, 191, 300), meta=np.ndarray>
Attributes:
    long_name:  vertical derivative of salinity
    time:       ocean_time
    field:      salinity, scalar, series
    name:       dsaltdz
    units:      1/m * units
```

Horizontal

Syntax:

```
ds.xroms.ddxi('u');  # horizontal xi-direction gradient (accessor)

ds.xroms.ddeta('u');  #  horizontal eta-direction gradient (accessor)

dtempdxi, dtempdeta = xroms.hgrad(ds.temp, xgrid)  # both gradients simultaneously, as␣
↪function

xroms.ddxi(ds.temp, xgrid)  # individual derivative, as function

xroms.ddeta(ds.temp, xgrid)  # individual derivative, as function
```

```
ds.xroms.ddxi('u')  # horizontal xi-direction gradient
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

(continues on next page)

(continued from previous page)

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
<xarray.DataArray 'dudxi' (ocean_time: 2, s_w: 31, eta_rho: 191, xi_rho: 300)> Size: 28MB
dask.array<sub, shape=(2, 31, 191, 300), dtype=float64, chunksize=(2, 31, 191, 300),␣
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_w         (s_w) float64 248B -1.0 -0.9667 -0.9333 ... -0.03333 0.0
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    z_w         (ocean_time, s_w, eta_rho, xi_rho) float64 28MB dask.array<chunksize=(2,␣
↪31, 191, 300), meta=np.ndarray>
Attributes:
```

(continues on next page)

(continued from previous page)

```
    long_name:  horizontal xi derivative of u-momentum component
    units:      1/m * meter second-1
    time:       ocean_time
    field:      u-velocity, scalar, series
    name:       dudxi
```

Time

Use `xarray` directly for this.

```
ddt = ds.differentiate('ocean_time', datetime_unit='s')
ddt
```

```
<xarray.Dataset> Size: 734MB
Dimensions:     (eta_rho: 191, xi_rho: 300, s_rho: 30, s_w: 31, ocean_time: 2,
                 xi_u: 299, eta_v: 190)
Coordinates: (12/21)
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_rho       (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * s_w         (s_w) float64 248B -1.0 -0.9667 -0.9333 ... -0.03333 0.0
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
    lon_u       (eta_rho, xi_u) float64 457kB dask.array<chunksize=(191, 299), meta=np.
↪ndarray>
    ...          ...
    z_w_v       (ocean_time, s_w, eta_v, xi_rho) float64 28MB dask.array<chunksize=(2,␣
↪31, 190, 300), meta=np.ndarray>
    z_w_psi     (ocean_time, s_w, eta_v, xi_u) float64 28MB dask.array<chunksize=(2, 31,␣
↪190, 299), meta=np.ndarray>
    z_rho       (ocean_time, s_rho, eta_rho, xi_rho) float64 28MB dask.array
↪<chunksize=(2, 30, 191, 300), meta=np.ndarray>
    z_rho_u     (ocean_time, s_rho, eta_rho, xi_u) float64 27MB dask.array<chunksize=(2,␣
↪30, 191, 299), meta=np.ndarray>
    z_rho_v     (ocean_time, s_rho, eta_v, xi_rho) float64 27MB dask.array<chunksize=(2,␣
↪30, 190, 300), meta=np.ndarray>
    z_rho_psi   (ocean_time, s_rho, eta_v, xi_u) float64 27MB dask.array<chunksize=(2,␣
↪30, 190, 299), meta=np.ndarray>
Data variables: (12/45)
    angle       (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    hc          float64 8B ...
    Cs_r        (s_rho) float64 240B dask.array<chunksize=(30,), meta=np.ndarray>
    zeta        (ocean_time, eta_rho, xi_rho) float32 458kB dask.array<chunksize=(2, 191,
↪ 300), meta=np.ndarray>
    h           (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    Cs_w        (s_w) float64 248B dask.array<chunksize=(31,), meta=np.ndarray>
    ...          ...
    dV_w_u      (ocean_time, s_w, eta_rho, xi_u) float64 28MB dask.array<chunksize=(2,␣
```

(continues on next page)

(continued from previous page)

```
↪31, 191, 299), meta=np.ndarray>
    dV_v        (ocean_time, s_rho, eta_v, xi_rho) float64 27MB dask.array<chunksize=(2,␣
↪30, 190, 300), meta=np.ndarray>
    dV_w_v      (ocean_time, s_w, eta_v, xi_rho) float64 28MB dask.array<chunksize=(2,␣
↪31, 190, 300), meta=np.ndarray>
    dV_psi      (ocean_time, s_rho, eta_v, xi_u) float64 27MB dask.array<chunksize=(2,␣
↪30, 190, 299), meta=np.ndarray>
    dV_w_psi    (ocean_time, s_w, eta_v, xi_u) float64 28MB dask.array<chunksize=(2, 31,␣
↪190, 299), meta=np.ndarray>
    rho0        int64 8B 1025
Attributes: (12/29)
    file:               ocean_his_0150.nc
    format:             netCDF-3 classic file
    Conventions:        CF-1.4
    type:               ROMS/TOMS history file
    title:              Texas and Louisiana Shelf case (Nesting)
    rst_file:           ocean_rst.nc
    ...                 ...
    compiler_command:   /g/software/openmpi/1.4.3/intel/bin/mpif90
    compiler_flags:     -heap-arrays -fp-model precise -assume 2underscores -c...
    tiling:             016x032
    history:            ROMS/TOMS, Version 3.4, Sunday - December 4, 2011 -  7...
    ana_file:           /scratch/zhangxq/projects/txla_nesting6/Functionals/an...
    CPP_options:        TXLA, ANA_BSFLUX, ANA_BTFLUX, ASSUMED_SHAPE, BULK_FLUX...
```

1.3.10 Built-in Physical Calculations

These are all properties of the accessor, so should be called without (). Demonstrated below are the calculations using the accessor and not using the accessor.

Horizontal speed

```
ds.xroms.speed  # accessor

xroms.speed(ds.u, ds.v, xgrid)  # function
```

```
ds.xroms.speed
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
```

(continues on next page)

(continued from previous page)

```
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
<xarray.DataArray 'speed' (ocean_time: 2, s_rho: 30, eta_rho: 191, xi_rho: 300)> Size:␣
↪14MB
dask.array<sqrt, shape=(2, 30, 191, 300), dtype=float32, chunksize=(2, 30, 191, 300),␣
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_rho       (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    z_rho       (ocean_time, s_rho, eta_rho, xi_rho) float64 28MB dask.array
↪<chunksize=(2, 30, 191, 300), meta=np.ndarray>
Attributes:
    long_name:  horizontal speed
    units:      m/s
    time:       ocean_time
    field:      u-velocity, scalar, series
    name:       speed
```

Kinetic energy

```
ds.xroms.KE  # accessor

# without the accessor you need to manage this yourself - first calculate speed to then␣
↪calculate KE
speed = xroms.speed(ds.u, ds.v, xgrid)
xroms.KE(ds.rho0, speed)
```

```
ds.xroms.KE
```

```
<xarray.DataArray 'KE' (ocean_time: 2, s_rho: 30, eta_rho: 191, xi_rho: 300)> Size: 28MB
dask.array<multiply, shape=(2, 30, 191, 300), dtype=float64, chunksize=(2, 30, 191, 300),
↪ chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_rho       (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    z_rho       (ocean_time, s_rho, eta_rho, xi_rho) float64 28MB dask.array
↪<chunksize=(2, 30, 191, 300), meta=np.ndarray>
```

(continues on next page)

(continued from previous page)

```
Attributes:
    name:       KE
    long_name:  kinetic energy
    units:      kg/(m*s^2)
```

Geostrophic velocities

```
ds.xroms.ug  # accessor, u component
ds.xroms.vg  # accessor, v component

ug, vg = xroms.uv_geostrophic(ds.zeta, ds.f, xgrid)  # function
```

```
ds.xroms.ug
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
<xarray.DataArray 'ug' (ocean_time: 2, eta_rho: 191, xi_u: 299)> Size: 914kB
dask.array<truediv, shape=(2, 191, 299), dtype=float64, chunksize=(2, 191, 299),␣
↪chunktype=numpy.ndarray>
Coordinates:
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
    lon_u       (eta_rho, xi_u) float64 457kB dask.array<chunksize=(191, 299), meta=np.
↪ndarray>
    lat_u       (eta_rho, xi_u) float64 457kB dask.array<chunksize=(191, 299), meta=np.
↪ndarray>
  * xi_u        (xi_u) int64 2kB 0 1 2 3 4 5 6 7 ... 292 293 294 295 296 297 298
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
```

(continues on next page)

(continued from previous page)

```
Attributes:
    long_name:  geostrophic u velocity
    units:      m/s
    time:       ocean_time
    field:      free-surface, scalar, series
    name:       ug
```

Eddy kinetic energy (EKE)

```
ds.xroms.EKE  # accessor

ug, vg = xroms.uv_geostrophic(ds.zeta, ds.f, xgrid)
xroms.EKE(ug, vg, xgrid)
```

```
ds.xroms.EKE
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
```

(continues on next page)

(continued from previous page)

```
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
<xarray.DataArray 'EKE' (ocean_time: 2, eta_rho: 191, xi_rho: 300)> Size: 917kB
dask.array<mul, shape=(2, 191, 300), dtype=float64, chunksize=(2, 191, 300),␣
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
Attributes:
    long_name:  eddy kinetic energy
    units:      m^2/s^2
    time:       ocean_time
    field:      free-surface, scalar, series
    name:       EKE
```

Vertical shear

Since it is a common use case, there are specific methods to return the u and v components of vertical shear on their own grids. These are just available for Datasets.

```
ds.xroms.dudz
ds.xroms.dvdz

xroms.dudz(ds.u, xgrid)
xroms.dvdz(ds.v, xgrid)

# already on same grid:
ds.xroms.vertical_shear

dudz = xroms.dudz(ds.u, xgrid)
dvdz = xroms.dvdz(ds.v, xgrid)
xroms.vertical_shear(dudz, dvdz, xgrid)
```

```
ds.xroms.dudz
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
<xarray.DataArray 'dudz' (ocean_time: 2, s_w: 31, eta_rho: 191, xi_u: 299)> Size: 28MB
dask.array<truediv, shape=(2, 31, 191, 299), dtype=float64, chunksize=(2, 31, 191, 299),␣
```

(continues on next page)

(continued from previous page)

```
↪chunktype=numpy.ndarray>
Coordinates:
  * s_w         (s_w) float64 248B -1.0 -0.9667 -0.9333 ... -0.03333 0.0
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
    lon_u       (eta_rho, xi_u) float64 457kB dask.array<chunksize=(191, 299), meta=np.
↪ndarray>
    lat_u       (eta_rho, xi_u) float64 457kB dask.array<chunksize=(191, 299), meta=np.
↪ndarray>
  * xi_u        (xi_u) int64 2kB 0 1 2 3 4 5 6 7 ... 292 293 294 295 296 297 298
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    z_w_u       (ocean_time, s_w, eta_rho, xi_u) float64 28MB dask.array<chunksize=(2,␣
↪31, 191, 299), meta=np.ndarray>
Attributes:
    name:       dudz
    long_name:  u component of vertical shear
    units:      1/s
```

The magnitude of the vertical shear is also a built-in derived variable for the `xroms` accessor:

```
ds.xroms.vertical_shear
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
<xarray.DataArray 'shear' (ocean_time: 2, s_w: 31, eta_rho: 191, xi_rho: 300)> Size: 28MB
dask.array<sqrt, shape=(2, 31, 191, 300), dtype=float64, chunksize=(2, 31, 191, 300),␣
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_w         (s_w) float64 248B -1.0 -0.9667 -0.9333 ... -0.03333 0.0
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
```

(continues on next page)

(continued from previous page)

```
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    z_w         (ocean_time, s_w, eta_rho, xi_rho) float64 28MB dask.array<chunksize=(2,␣
↪31, 191, 300), meta=np.ndarray>
Attributes:
    name:       shear
    long_name:  vertical shear
    units:      1/s
```

Vertical vorticity

```
ds.xroms.vort

xroms.relative_vorticity(ds.u, ds.v, xgrid)
```

```
ds.xroms.vort
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
```

(continues on next page)

(continued from previous page)

```
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

(continues on next page)

(continued from previous page)

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
<xarray.DataArray 'vort' (ocean_time: 2, s_w: 31, eta_v: 190, xi_u: 299)> Size: 28MB
dask.array<sub, shape=(2, 31, 190, 299), dtype=float64, chunksize=(2, 31, 190, 299),␣
↪chunktype=numpy.ndarray>
Coordinates:
  * s_w         (s_w) float64 248B -1.0 -0.9667 -0.9333 ... -0.03333 0.0
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_u        (xi_u) int64 2kB 0 1 2 3 4 5 6 7 ... 292 293 294 295 296 297 298
  * eta_v       (eta_v) int64 2kB 0 1 2 3 4 5 6 ... 183 184 185 186 187 188 189
    z_w_psi     (ocean_time, s_w, eta_v, xi_u) float64 28MB dask.array<chunksize=(2, 31,␣
↪190, 299), meta=np.ndarray>
Attributes:
    long_name:  vertical component of vorticity
    units:      1/s
    time:       ocean_time
    field:      v-velocity, scalar, series
    name:       vort
```

Horizontal convergence

Horizontal component of the currents convergence.

```
ds.xroms.convergence

xroms.convergence(ds.u, ds.v, xgrid)
```

```
ds.xroms.convergence
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
```

(continues on next page)

(continued from previous page)

```
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
```

(continues on next page)

(continued from previous page)

```
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
<xarray.DataArray 'convergence' (ocean_time: 2, s_rho: 30, eta_rho: 191,
                                 xi_rho: 300)> Size: 28MB
```

(continues on next page)

(continued from previous page)

```
dask.array<add, shape=(2, 30, 191, 300), dtype=float64, chunksize=(2, 30, 191, 300),␣
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_rho       (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    z_rho       (ocean_time, s_rho, eta_rho, xi_rho) float64 28MB dask.array
↪<chunksize=(2, 30, 191, 300), meta=np.ndarray>
Attributes:
    long_name:  horizontal convergence
    units:      1/s
    time:       ocean_time
    field:      u-velocity, scalar, series
    name:       convergence
```

Normalized surface convergence

Horizontal component of the currents convergence at the surface, normalized by f. This is only available through the accessor.

```
ds.xroms.convergence_norm
```

```
ds.xroms.convergence_norm
```

```
<xarray.DataArray 'convergence_norm' (ocean_time: 2, eta_rho: 191, xi_rho: 300)> Size:␣
↪917kB
dask.array<truediv, shape=(2, 191, 300), dtype=float64, chunksize=(2, 191, 300),␣
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
Attributes:
    name:       convergence_norm
    long_name:  normalized surface horizontal convergence
    units:
```

Ertel potential vorticity

The accessor assumes you want the Ertel potential vorticity of the buoyancy:

```
ds.xroms.ertel

sig0 = xroms.potential_density(ds.temp, ds.salt)
buoyancy = xroms.buoyancy(sig0, rho0=ds.rho0)
xroms.ertel(buoyancy, ds.u, ds.v, ds.f, xgrid, scoord='s_w')
```

Alternatively, the user can access the original function and use a different tracer for this calculation (in this example, "dye_01"), and can return the result on a different vertical grid, for example:

```
xroms.ertel(ds.dye_01, ds.u, ds.v, ds.f, xgrid, scoord='s_w')
```

```
ds.xroms.ertel
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

(continues on next page)

(continued from previous page)

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
```

(continues on next page)

(continued from previous page)

```
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
```

(continues on next page)

(continued from previous page)

```
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
```

(continues on next page)

(continued from previous page)

```
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
```

(continues on next page)

(continued from previous page)

```
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
<xarray.DataArray 'ertel' (ocean_time: 2, s_rho: 30, eta_rho: 191, xi_rho: 300)> Size:␣
↪28MB
dask.array<add, shape=(2, 30, 191, 300), dtype=float64, chunksize=(2, 30, 191, 300),␣
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_rho       (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
```

(continues on next page)

(continued from previous page)

```
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    z_rho       (ocean_time, s_rho, eta_rho, xi_rho) float64 28MB dask.array
↪<chunksize=(2, 30, 191, 300), meta=np.ndarray>
Attributes:
    name:       ertel
    long_name:  ertel potential vorticity
    units:      tracer/(m*s)
```

Density

```
ds.xroms.rho

xroms.density(ds.temp, ds.salt)
```

```
ds.xroms.rho
```

```
<xarray.DataArray 'rho' (ocean_time: 2, s_rho: 30, eta_rho: 191, xi_rho: 300)> Size: 28MB
dask.array<truediv, shape=(2, 30, 191, 300), dtype=float64, chunksize=(2, 30, 191, 300),␣
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_rho       (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    z_rho       (ocean_time, s_rho, eta_rho, xi_rho) float64 28MB dask.array
↪<chunksize=(2, 30, 191, 300), meta=np.ndarray>
Attributes:
    long_name:  density
    units:      kg/m^3
    time:       ocean_time
    field:      temperature, scalar, series
    name:       rho
```

Potential density

```
ds.xroms.sig0

xroms.potential_density(ds.temp, ds.salt)
```

```
ds.xroms.sig0
```

```
<xarray.DataArray 'sig0' (ocean_time: 2, s_rho: 30, eta_rho: 191, xi_rho: 300)> Size:␣
↪14MB
dask.array<truediv, shape=(2, 30, 191, 300), dtype=float32, chunksize=(2, 30, 191, 300),␣
```

(continues on next page)

(continued from previous page)

```
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_rho       (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    z_rho       (ocean_time, s_rho, eta_rho, xi_rho) float64 28MB dask.array
↪<chunksize=(2, 30, 191, 300), meta=np.ndarray>
Attributes:
    long_name:  potential density
    units:      kg/m^3
    time:       ocean_time
    field:      temperature, scalar, series
    name:       sig0
```

Buoyancy

```
ds.xroms.buoyancy

sig0 = xroms.potential_density(ds.temp, ds.salt);
xroms.buoyancy(sig0)
```

```
ds.xroms.buoyancy
```

```
<xarray.DataArray 'buoyancy' (ocean_time: 2, s_rho: 30, eta_rho: 191,
                              xi_rho: 300)> Size: 28MB
dask.array<truediv, shape=(2, 30, 191, 300), dtype=float64, chunksize=(2, 30, 191, 300),␣
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_rho       (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    z_rho       (ocean_time, s_rho, eta_rho, xi_rho) float64 28MB dask.array
↪<chunksize=(2, 30, 191, 300), meta=np.ndarray>
Attributes:
    long_name:  buoyancy
    units:      m/s^2
    time:       ocean_time
    field:      temperature, scalar, series
    name:       buoyancy
```

Buoyancy frequency

Also called vertical buoyancy gradient.

```
ds.xroms.N2

rho = xroms.density(ds.temp, ds.salt)  # calculate rho if not in output
xroms.N2(rho, xgrid)
```

```
ds.xroms.N2
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
<xarray.DataArray 'N2' (ocean_time: 2, s_w: 31, eta_rho: 191, xi_rho: 300)> Size: 28MB
dask.array<truediv, shape=(2, 31, 191, 300), dtype=float64, chunksize=(2, 31, 191, 300),␣
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_w         (s_w) float64 248B -1.0 -0.9667 -0.9333 ... -0.03333 0.0
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    z_w         (ocean_time, s_w, eta_rho, xi_rho) float64 28MB dask.array<chunksize=(2,␣
↪31, 191, 300), meta=np.ndarray>
Attributes:
    long_name:  buoyancy frequency squared, or vertical buoyancy gradient
    units:      1/s^2
    time:       ocean_time
    field:      temperature, scalar, series
    name:       N2
```

Horizontal buoyancy gradient

```
ds.xroms.M2

rho = xroms.density(ds.temp, ds.salt)  # calculate rho if not in output
xroms.M2(rho, xgrid)
```

```
ds.xroms.M2
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
```

(continues on next page)

(continued from previous page)

```
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

(continues on next page)

(continued from previous page)

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
```

(continues on next page)

(continued from previous page)

```
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
<xarray.DataArray 'M2' (ocean_time: 2, s_w: 31, eta_rho: 191, xi_rho: 300)> Size: 28MB
dask.array<truediv, shape=(2, 31, 191, 300), dtype=float64, chunksize=(2, 31, 191, 300),␣
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_w         (s_w) float64 248B -1.0 -0.9667 -0.9333 ... -0.03333 0.0
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    z_w         (ocean_time, s_w, eta_rho, xi_rho) float64 28MB dask.array<chunksize=(2,␣
↪31, 191, 300), meta=np.ndarray>
Attributes:
    long_name:  horizontal buoyancy gradient
    units:      1/s^2
    time:       ocean_time
    field:      temperature, scalar, series
    name:       M2
```

Mixed layer depth

This is not a property since the threshold is a parameter and needs to be input.

```
ds.xroms.mld(thresh=0.03)

sig0 = xroms.potential_density(ds.temp, ds.salt);
xroms.mld(sig0, xgrid, ds.h, ds.mask_rho, thresh=0.03)
```

```
ds.xroms.mld(thresh=0.03)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid.py:989: FutureWarning: From version 0.8.0 the Axis computation␣
↪methods will be removed, in favour of using the Grid computation methods instead. i.e.␣
↪use `Grid.transform` instead of `Axis.transform`
  warnings.warn(
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid.py:989: FutureWarning: From version 0.8.0 the Axis computation␣
↪methods will be removed, in favour of using the Grid computation methods instead. i.e.␣
↪use `Grid.transform` instead of `Axis.transform`
  warnings.warn(
```

```
<xarray.DataArray 'mld' (ocean_time: 2, eta_rho: 191, xi_rho: 300)> Size: 917kB
dask.array<abs, shape=(2, 191, 300), dtype=float64, chunksize=(2, 191, 300),␣
↪chunktype=numpy.ndarray>
Coordinates:
```

(continues on next page)

(continued from previous page)

```
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
    sig0        float64 8B 0.0
    z_rho       (ocean_time, eta_rho, xi_rho) float64 917kB dask.array<chunksize=(2, 191,
↪ 300), meta=np.ndarray>
Attributes:
    long_name:      mixed layer depth
    time:           ocean_time
    field:          z_rho, scalar, series
    units:          m
    positive:       up
    standard_name:  depth
    name:           mld
```

1.3.11 Other calculations

Rotations

If your ROMS grid is curvilinear, you'll need to rotate your u and v velocities from along the grid axes to being eastward and northward. You can do this with

```
ds.xroms.east

ds.xroms.north
```

Additionally, if you want to rotate your velocity to be a different orientation, for example to be along-channel, you can do that with

```
ds.xroms.east_rotated(angle)

ds.xroms.north_rotated(angle)
```

1.3.12 Time-based calculations including climatologies

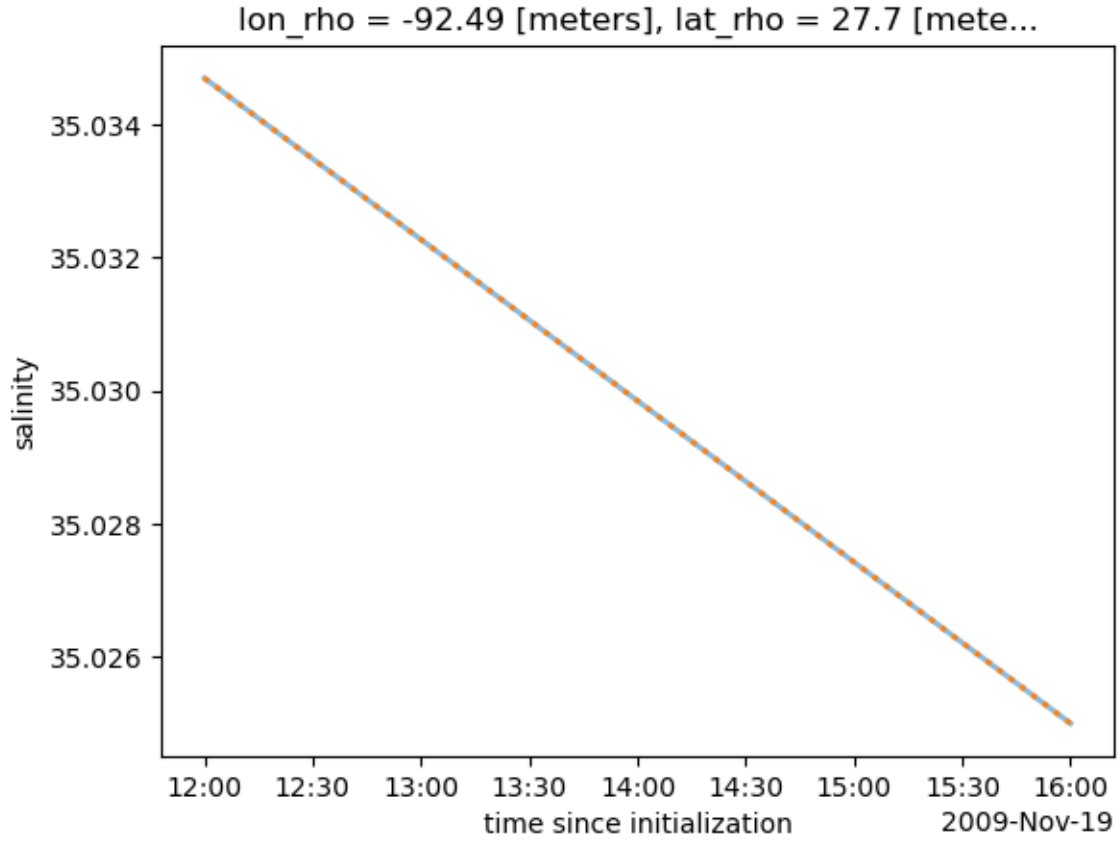
Rolling averages in time

Here is an example of computing a rolling average in time. Nothing happens in this example because we only have two time steps to use, however, it does demonstrate the syntax. If more time steps were available we would update `ds.salt.rolling(ocean_time=1)` to include more time steps to average over in a rolling sense.

More information about rolling operations is available.

```
roll = ds.salt.rolling(ocean_time=1, center=True, min_periods=1).mean()
roll.isel(s_rho=-1, eta_rho=10, xi_rho=20).plot(alpha=0.5, lw=2)
ds.salt.isel(s_rho=-1, eta_rho=10, xi_rho=20).plot(ls=":", lw=2)
```

```
[<matplotlib.lines.Line2D at 0x7f93b8594d00>]
```



Resampling in time

Can't have any chunks in the time dimension to do this. More info: http://xarray.pydata.org/en/stable/generated/xarray.Dataset.resample.html

Upsample

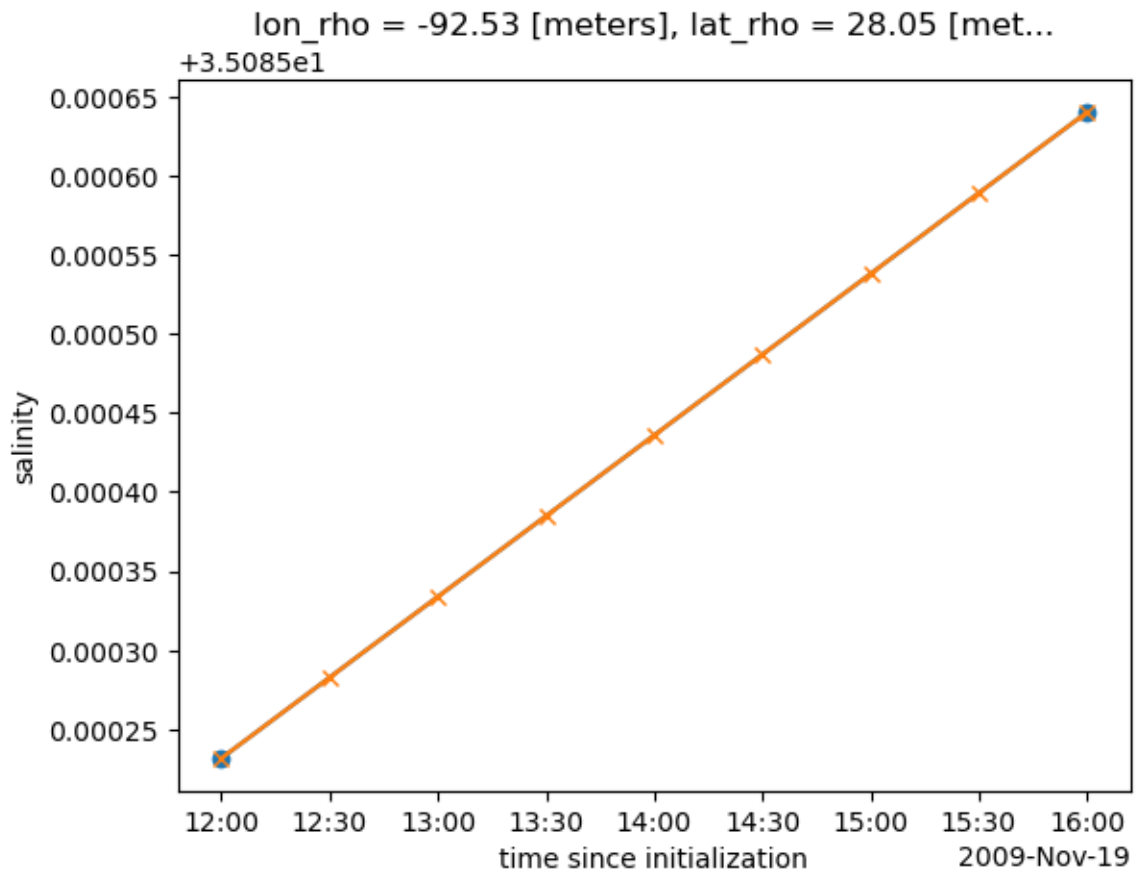
Upsample to a higher resolution in time. Makes sense to interpolate to fill in data when upsampling, but can also forward or backfill, or just add nan's.

```
dstest = ds.resample(ocean_time='30min', restore_coord_dims=True).interpolate()
```

Plot to visually inspect results

```
ds.salt.cf.isel(Y=30, X=20, Z=-1).plot(marker='o')
dstest.salt.cf.isel(Y=30, X=20, Z=-1).plot(marker='x')
```

```
[<matplotlib.lines.Line2D at 0x7f93b82d6290>]
```



Downsample

Resample down to lower resolution in time. This requires appending a method to aggregate the extra data, such as a `mean`. Note that other options can be used to shift the result within the interval of aggregation in various ways. Just the syntax is shown here since we only have two time steps to work with.

```
dstest = ds.resample(ocean_time='6H').mean()
ds.salt.cf.isel(Y=30, X=20, Z=-1).plot(marker='o')
dstest.salt.cf.isel(Y=30, X=20, Z=-1).plot(marker='x')
```

Seasonal average, over time

This is an example of resampling.

```
da.cf.resample({'T': [time frequency string]}).reduce([aggregation function])
```

For example, calculate the mean temperature every quarter in time with the following:

```
ds.temp.cf.resample({'T': 'QS'}).reduce(np.mean)
```

or the aggregation function can be appended on the end directly with:

```
ds.temp.cf.resample({'T': 'QS'}).mean()
```

The result of this calculation is a time series of downsampled chunks of output in time, the frequency of which is selected by input "time frequency string", and aggregated by input "aggregation function".

Examples of the time frequency string are:

- "QS": quarters, starting in January of each year and averaging three months.
  - Also available are selections like "QS-DEC", quarters but starting with December to better align with seasons. Other months are input options as well.
- "MS": monthly
- "D": daily
- "H": hourly
- Many more options are given here.

Examples of aggregation functions are:

- np.mean
- np.max
- np.min
- np.sum
- np.std

Result of downsampling a 4D salt array from hourly to 6-hourly, for example, gives: `[ocean_time x s_rho x eta_rho x xi_rho]`, where `ocean_time` has about 1/6 of the number of entries reflecting the aggregation in time.

```
ds.temp.cf.resample(indexer={'T': '6H'}).reduce(np.mean)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xarray/core/groupby.py:532: FutureWarning: 'H' is deprecated and will be␣
↪removed in a future version, please use 'h' instead.
  index_grouper = pd.Grouper(
```

```
<xarray.DataArray 'temp' (ocean_time: 1, s_rho: 30, eta_rho: 191, xi_rho: 300)> Size: 7MB
dask.array<stack, shape=(1, 30, 191, 300), dtype=float32, chunksize=(1, 30, 191, 300),␣
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_rho       (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * xi_rho      (xi_rho) int64 2kB 0 1 2 3 4 5 6 ... 293 294 295 296 297 298 299
  * eta_rho     (eta_rho) int64 2kB 0 1 2 3 4 5 6 ... 185 186 187 188 189 190
  * ocean_time  (ocean_time) datetime64[ns] 8B 2009-11-19T12:00:00
Attributes:
    long_name:  potential temperature
    units:      Celsius
    time:       ocean_time
    field:      temperature, scalar, series
```

Seasonal mean over all available time

This is how to average over the full dataset period by certain time groupings using xarray `groupby` which is like pandas version. In this case we show the seasonal mean averaged across the full model time period. The syntax for this is:

```
da.salt.cf.groupby('T.[time string]').reduce([aggregation function])
```

For example, to average salt by season:

```
da.salt.cf.groupby('T.season').reduce(np.mean)
```

or

```
da.salt.cf.groupby('T.season').mean()
```

Options for the time string include:

- 'season'
- 'year'
- 'month'
- 'day'
- 'hour'
- 'minute'
- 'second'
- 'dayofyear'
- 'week'
- 'dayofweek'
- 'weekday'
- 'quarter'

More information about options for time (including "derived" datetime coordinates) is here.

Examples of aggregation functions are:

- np.mean
- np.max
- np.min
- np.sum
- np.std

Result of averaging over seasons for a 4D salt array returns, for example: `[season x s_rho x eta_rho x xi_rho]`, where `season` has 4 entries, each covering 3 months of the year.

```
# this example has only 1 season because it is a short example file
ds.temp.cf.groupby('T.season').mean()
```

```
<xarray.DataArray 'temp' (season: 1, s_rho: 30, eta_rho: 191, xi_rho: 300)> Size: 7MB
dask.array<stack, shape=(1, 30, 191, 300), dtype=float32, chunksize=(1, 30, 191, 300),␣
↪chunktype=numpy.ndarray>
Coordinates:
    lon_rho  (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho  (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_rho    (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * xi_rho   (xi_rho) int64 2kB 0 1 2 3 4 5 6 7 ... 293 294 295 296 297 298 299
  * eta_rho  (eta_rho) int64 2kB 0 1 2 3 4 5 6 7 ... 184 185 186 187 188 189 190
  * season   (season) object 8B 'SON'
Attributes:
    long_name:  potential temperature
    units:      Celsius
    time:       ocean_time
    field:      temperature, scalar, series
```

```
import xarray as xr
import xroms
import numpy as np
import matplotlib.pyplot as plt
import cmocean.cm as cmo
import pandas as pd
```

1.4 How to interpolate

There is a different approach for interpolation in:

- time (using `xarray interp`)
- longitude/latitude (using `xESMF`)
- depth (using `xgcm`)

In this notebook, we will demonstrate each independently as well considerations for combining the approaches.

1.4.1 Load in data

Load in example dataset. More information at in *input/output page*

```
ds = xroms.datasets.fetch_ROMS_example_full_grid()
ds, xgrid = xroms.roms_dataset(ds, include_cell_volume=True, include_Z0=True)
ds.xroms.set_grid(xgrid)
ds
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
```

(continues on next page)

(continued from previous page)

```
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
```

(continues on next page)

(continued from previous page)

```
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
```

(continues on next page)

(continued from previous page)

```
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
```

(continues on next page)

(continued from previous page)

```
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
```

(continues on next page)

(continued from previous page)

```
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
```

(continues on next page)

(continued from previous page)

```
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
```

(continues on next page)

(continued from previous page)

```
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
<xarray.Dataset> Size: 957MB
Dimensions:     (eta_rho: 191, xi_rho: 300, s_rho: 30, s_w: 31, ocean_time: 2,
                 xi_u: 299, eta_v: 190)
Coordinates: (12/29)
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_rho       (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * s_w         (s_w) float64 248B -1.0 -0.9667 -0.9333 ... -0.03333 0.0
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
    lon_u       (eta_rho, xi_u) float64 457kB dask.array<chunksize=(191, 299), meta=np.
↪ndarray>
    ...          ...
    z_rho_v0    (s_rho, eta_v, xi_rho) float64 14MB dask.array<chunksize=(30, 190, 300),␣
↪meta=np.ndarray>
```

(continues on next page)

(continued from previous page)

```
    z_rho_psi0  (s_rho, eta_v, xi_u) float64 14MB dask.array<chunksize=(30, 190, 299),␣
↪meta=np.ndarray>
    z_w0        (s_w, eta_rho, xi_rho) float64 14MB dask.array<chunksize=(31, 191, 300),␣
↪meta=np.ndarray>
    z_w_u0      (s_w, eta_rho, xi_u) float64 14MB dask.array<chunksize=(31, 191, 299),␣
↪meta=np.ndarray>
    z_w_v0      (s_w, eta_v, xi_rho) float64 14MB dask.array<chunksize=(31, 190, 300),␣
↪meta=np.ndarray>
    z_w_psi0    (s_w, eta_v, xi_u) float64 14MB dask.array<chunksize=(31, 190, 299),␣
↪meta=np.ndarray>
Data variables: (12/53)
    angle       (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    hc          float64 8B ...
    Cs_r        (s_rho) float64 240B dask.array<chunksize=(30,), meta=np.ndarray>
    zeta        (ocean_time, eta_rho, xi_rho) float32 458kB dask.array<chunksize=(2, 191,
↪ 300), meta=np.ndarray>
    h           (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    Cs_w        (s_w) float64 248B dask.array<chunksize=(31,), meta=np.ndarray>
    ...          ...
    dV_w_u      (ocean_time, s_w, eta_rho, xi_u) float64 28MB dask.array<chunksize=(2,␣
↪31, 191, 299), meta=np.ndarray>
    dV_v        (ocean_time, s_rho, eta_v, xi_rho) float64 27MB dask.array<chunksize=(2,␣
↪30, 190, 300), meta=np.ndarray>
    dV_w_v      (ocean_time, s_w, eta_v, xi_rho) float64 28MB dask.array<chunksize=(2,␣
↪31, 190, 300), meta=np.ndarray>
    dV_psi      (ocean_time, s_rho, eta_v, xi_u) float64 27MB dask.array<chunksize=(2,␣
↪30, 190, 299), meta=np.ndarray>
    dV_w_psi    (ocean_time, s_w, eta_v, xi_u) float64 28MB dask.array<chunksize=(2, 31,␣
↪190, 299), meta=np.ndarray>
    rho0        int64 8B 1025
Attributes: (12/29)
    file:               ocean_his_0150.nc
    format:             netCDF-3 classic file
    Conventions:        CF-1.4
    type:               ROMS/TOMS history file
    title:              Texas and Louisiana Shelf case (Nesting)
    rst_file:           ocean_rst.nc
    ...                 ...
    compiler_command:   /g/software/openmpi/1.4.3/intel/bin/mpif90
    compiler_flags:     -heap-arrays -fp-model precise -assume 2underscores -c...
    tiling:             016x032
    history:            ROMS/TOMS, Version 3.4, Sunday - December 4, 2011 -  7...
    ana_file:           /scratch/zhangxq/projects/txla_nesting6/Functionals/an...
    CPP_options:        TXLA, ANA_BSFLUX, ANA_BTFLUX, ASSUMED_SHAPE, BULK_FLUX...
```

1.4.2 Interpolate to...

The following section is examples of different kinds of interpolation.

times

Interpolating in time is straight-forward because it is 1D, uncoupled from the other dimensions. So, we can just use the `xarray interp` function directly with the desired times. The result is `[ocean_time x s_rho x eta x xi]`.

Notes:

- The potentially tricky part is that chunking cannot occur in the direction of interpolation. So, here we reset the chunking and chunk in a different dimension before interpolation, then chunk back to `ocean_time` afterward.
- You can interpolate in time on the whole Dataset or a single DataArray. The example shows interpolation in time on a single DataArray.
- The interpolation times can be sequence-like, but I recommend putting them into a DataArray as follows and demonstrated in the example so that attributes are appropriately added and `cf-xarray` works afterward (also used in the other interpolation routines).

  t0s = xr.DataArray(t0s, dims='ocean_time', attrs={'axis': 'T', 'standard_name': 'time'})

Example usage for a DataArray `da`:

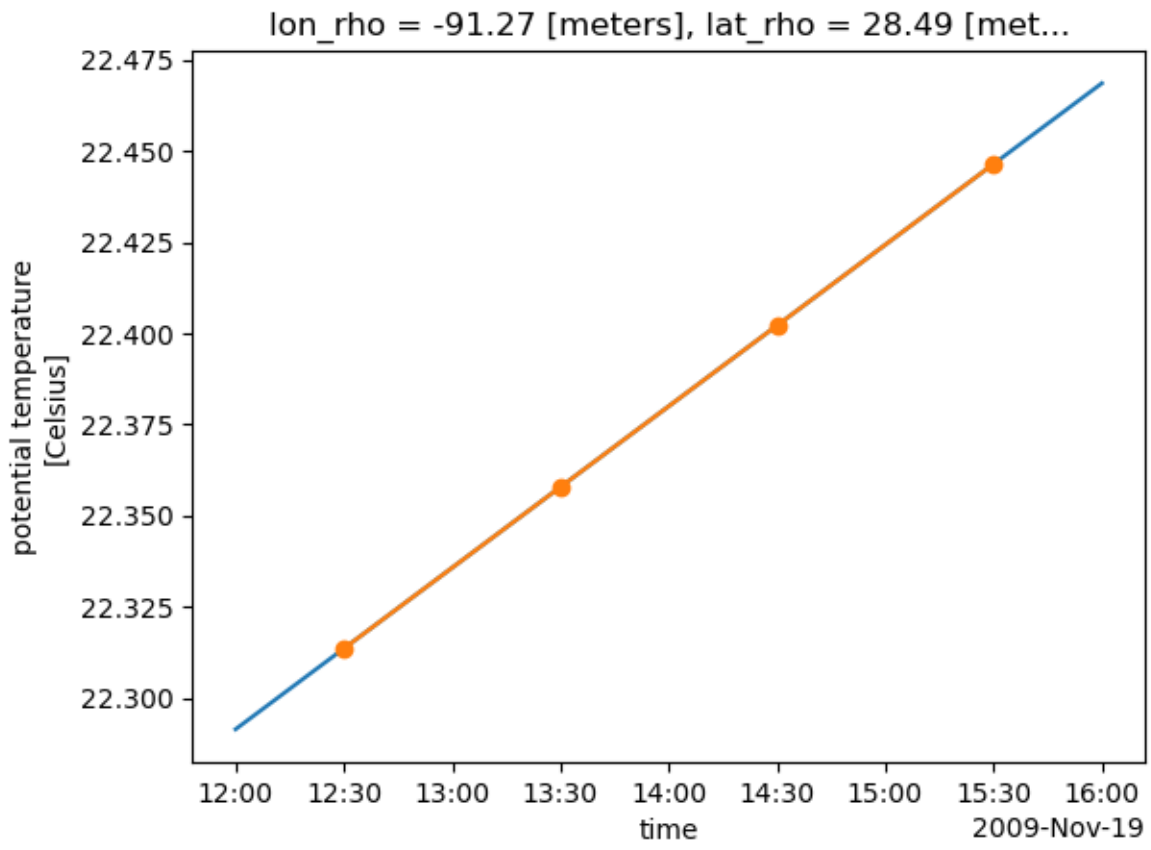
  da.interp(ocean_time=t0s)

```
# times to interpolate to
startdate = pd.Timestamp(ds.cf["T"][0].values)
t0s = [startdate + pd.Timedelta('30 min') + pd.Timedelta('1 hours')*i for i in range(4)]

# not necessary to change t0s to be a DataArray, but then we can add
# attributes that keep cf-xarray working
t0s = xr.DataArray(t0s, dims=ds.cf["T"].name, attrs={'axis': 'T', 'standard_name': 'time
↪'})

varin = ds.temp

# rechunk from time to vertical dimension
varin = varin.chunk({'ocean_time': -1, 's_rho': 1})

# interpolation
varout = varin.interp(ocean_time=t0s).chunk({'ocean_time':1, 's_rho': -1})
```

Results are demonstrated below for a single location.

```
varin.cf.isel(Z=-1, Y=50, X=100).plot()
varout.cf.isel(Z=-1, Y=50, X=100).plot(marker='o')
```

```
[<matplotlib.lines.Line2D at 0x7f254c1756f0>]
```



multiple lon, lat locations (1D)

Function `xroms.interpll` wraps `xESMF` so that the wrapper can take care of some niceties. It takes in longitude/latitude values and interpolates a variable onto the desired lon/lat positions correctly for a non-flat Earth. It has functionality for returning pairs of points (1D) vs. 2D arrays of points. First we demo the 1D output.

The result is dimensions `[ocean_time x s_rho x locations]`.

Notes:

- Cannot have chunks in the horizontal dimensions.
- 1D behavior is the default for `xroms.interpll` but also accessible by inputting `which='pairs'`.
- Input longitude and latitudes (below `lon0` and `lat0`) can be lists or ndarrays.

Example usage for a DataArray `da`:

```
xroms.interpll(da, lon0, lat0, which='pairs')
```

or with `xroms` accessor:

```
da.xroms.interpll(lon0, lat0, which='pairs')
```

```
# use advanced indexing to pull out individual pairs of points to compare with
# rather than 2D array of lon/lat points that would occur otherwise
ie, ix = [24, 100, 121, 30], [31, 198, 239, 142]
varin = ds.salt
indexer = {varin.cf["X"].name: xr.DataArray(ix, dims="locations"), varin.cf["Y"].name:␣
```

(continues on next page)

(continued from previous page)

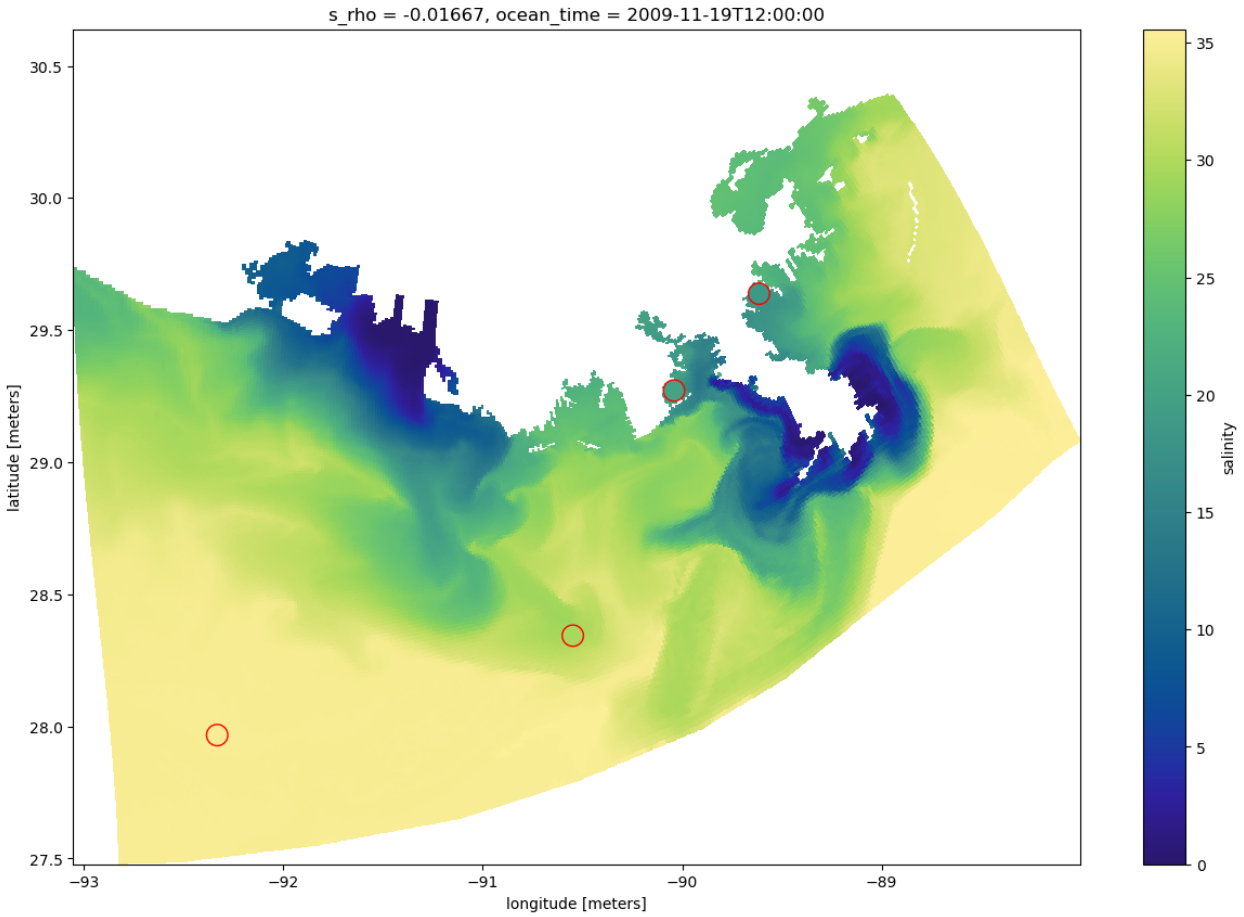
```
↪xr.DataArray(ie, dims="locations")}
lat0 = varin.cf["latitude"].isel(indexer)
lon0 = varin.cf["longitude"].isel(indexer)
varcomp = varin.isel(indexer).cf.isel(T=0, Z=-1)
```

```
varout = xroms.interpll(varin, lon0, lat0, which='pairs')
assert np.allclose(varout.isel(ocean_time=0, s_rho=-1), varcomp)
```

Plot the interpolated surface salinity overlaid on the full field to visually check.

```
indexer = {'ocean_time': 0, 's_rho': -1}
salt = varin.isel(indexer)
vmin = salt.min().values; vmax = salt.max().values

fig, ax = plt.subplots(1, 1, figsize=(15,10))
salt.cf.plot.pcolormesh(x='longitude', y='latitude', infer_intervals=True, cmap=cmo.
↪haline)
ax.scatter(lon0, lat0, c=varout.isel(indexer), s=200, edgecolor='r', vmin=vmin,␣
↪vmax=vmax, cmap=cmo.haline)
```

```
<matplotlib.collections.PathCollection at 0x7f253f892410>
```



array of lon, lat locations (2D)

We can also use `xroms.interpll` to interpolate to a 2D grid of longitudes and latitudes.

Result is `[ocean_time x s_rho x lat x lon]`.

Notes:

- Cannot have chunks in the horizontal dimensions.
- 2D grids of lon0, lat0 are found by inputting `which='grid'`.
- Input longitude and latitudes (below `lon0` and `lat0`) can be lists or ndarrays.

Example usage for a DataArray `da`:

    xroms.interpll(da, lon0, lat0, which='grid')

or with `xroms` accessor:

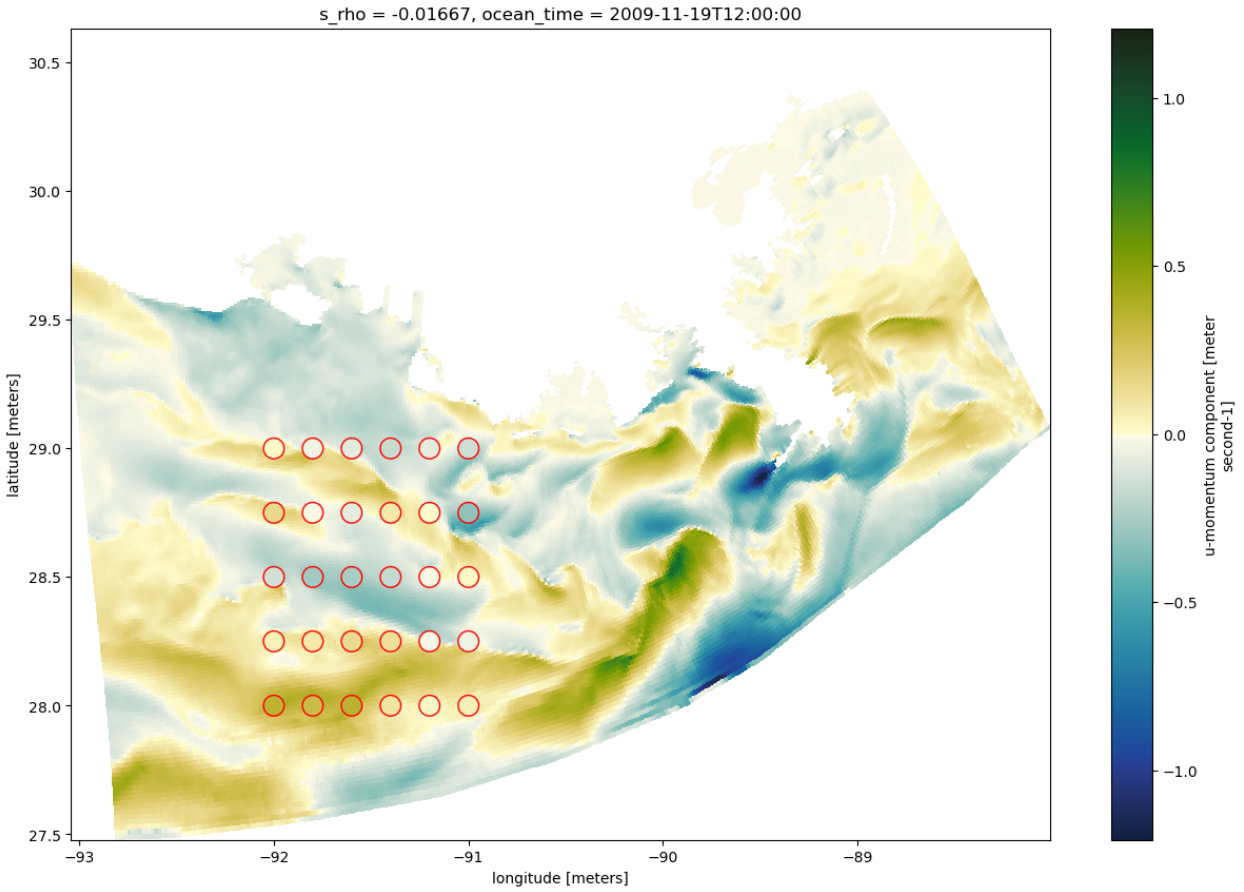
    da.xroms.interpll(lon0, lat0, which='grid')

```
npts = 5
lon0, lat0 = np.linspace(-92, -91, npts+1), np.linspace(28,29,npts)  # still input as 1D␣
↪arrays
LON0, LAT0 = np.meshgrid(lon0, lat0)  # for plotting
varin = ds.u

varout = xroms.interpll(varin, lon0, lat0, which='grid')
```

Plot to visually inspect results.

```
indexer = {'ocean_time': 0, 's_rho': -1}
vmin = abs(varin).min().values; vmax = abs(varin).max().values
vmax = max(vmin, vmax)

fig, ax = plt.subplots(1, 1, figsize=(15,10))
varin.isel(indexer).cf.plot.pcolormesh(x='longitude', y='latitude', infer_intervals=True,
↪ cmap=cmo.delta)
ax.scatter(LON0, LAT0, c=varout.isel(indexer), s=200, edgecolor='r', vmin=-vmax,␣
↪vmax=vmax, cmap=cmo.delta)
```

```
<matplotlib.collections.PathCollection at 0x7f253f753fa0>
```



variable regridded to fixed depths

Function `xroms.zslice` wraps `xgcm grid.transform` so that the wrapper can take care of some niceties. It interpolates a variable onto the input depths.

The result is dimensions `[ocean_time x [z coord] x eta x xi]`, where `[z coord]` is the z coordinate used to interpolate the variable to.

Notes:

- Cannot have chunks in the vertical dimension.
- Input depths can be lists or ndarrays.
- `xgcm grid.transform` has more flexibility and functionality than is offered through `xroms.zslice`; this function focuses on just depth interpolation.
- Interpolation to fixed depths can be done using time-varying depths or with constant depths in time; do the latter to save computation time if accuracy isn't very important.

with z varying in time

Use the z coordinates associated with the DataArray in the interpolation.

Example usage for a DataArray `da`:

> xroms.isoslice(da, depths, grid, z=z, axis="Z")

More is pre-selected if you used the `xroms` accessor, with a different name of "zslice". With DataArray, need to provide grid:

> da.xroms.zslice(grid, depths)

With Dataset accessor need to provide DataArray name:

> ds.xroms.zslice(varname, depths)

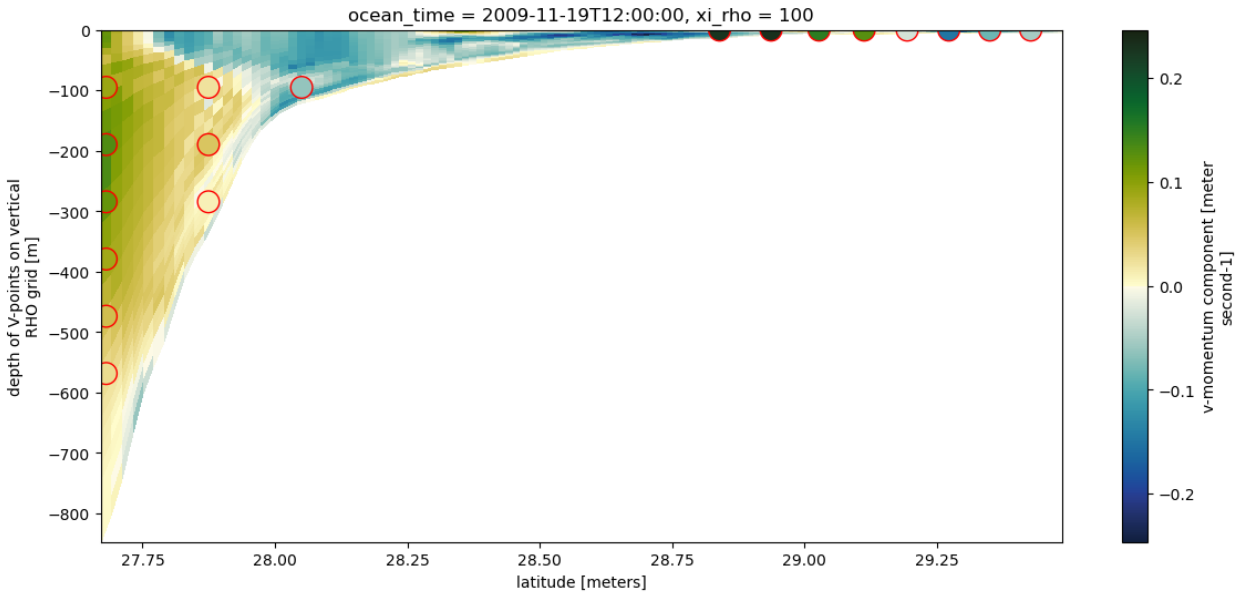
```
varin = ds.v
varout = xroms.isoslice(varin, np.linspace(0, -600, 20), xgrid)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid.py:989: FutureWarning: From version 0.8.0 the Axis computation␣
↪methods will be removed, in favour of using the Grid computation methods instead. i.e.␣
↪use `Grid.transform` instead of `Axis.transform`
  warnings.warn(
```

Plot to visually inspect results:

```
fig, ax = plt.subplots(1, 1, figsize=(14,6))

dss = varin.cf.isel(X=100, ocean_time=0)
dss.where(~dss.isnull().compute(), drop=True).cf.plot(x='latitude', y='vertical',␣
↪cmap=cmo.delta)

vmin = abs(dss).min().values; vmax = abs(dss).max().values
vmax = max(vmin, vmax)
toplot = varout.cf.isel(T=0, X=100, Y=slice(None,None,10), Z=slice(None,None,3))
X, Z = np.meshgrid(toplot.lat_v, toplot.z_rho_v)
ax.scatter(X, Z, c=toplot, s=200, edgecolor='r', vmin=-vmax, vmax=vmax, cmap=cmo.delta)
```

```
<matplotlib.collections.PathCollection at 0x7f253f79aa40>
```



z constant in time

Input separate z coordinates `z0` that don't vary in time for the DataArray to be interpolated to.

Example usage for a DataArray `da`:

> xroms.isoslice(da, depths, grid, z=z0, axis="Z")

More is pre-selected if you used the `xroms` accessor, with a different name of "zslice". With DataArray, need to provide grid:

> da.xroms.zslice(grid, depths, z=z0)

With Dataset accessor need to provide DataArray name:

> ds.xroms.zslice(varname, depths, z=z0)

One complication that is currently necessary is to change the metadata such that `z_rho_v0` is recognized as the vertical coordinate for `ds.v`.

```
var0 = ds.v

# changes to use z_rho_v0 as vertical coordinate
var0.attrs["coordinates"] = var0.attrs["coordinates"].replace("z_rho_v","z_rho_v0")
var0.z_rho_v0.attrs["positive"] = "up"
var0.z_rho_v0.attrs["standard_name"] = "depth"

varout0 = xroms.isoslice(var0, np.linspace(0, -600, 20), xgrid, iso_array=var0.z_rho_v0)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[12], line 4
      1 var0 = ds.v
      3 # changes to use z_rho_v0 as vertical coordinate
----> 4 var0.attrs["coordinates"] = var0.attrs["coordinates"].replace("z_rho_v","z_rho_v0
```

(continues on next page)

(continued from previous page)

```
↪")
      5 var0.z_rho_v0.attrs["positive"] = "up"
      6 var0.z_rho_v0.attrs["standard_name"] = "depth"

KeyError: 'coordinates'
```

Plot the difference between the two interpolations as a point to see the difference in accounting for time-varying depths and not.

```
indexer = {'ocean_time': 0, 'Y': 10, 'X': 250}

varout.cf.isel(indexer).cf.plot(y='vertical', figsize=(6,6), lw=3)
varout0.cf.isel(indexer).cf.plot(y='vertical')
```

multiple locations, depths, and times

A user can simply use multiple of these approaches one after another to interpolate in more dimensions. There are several considerations for the ordering:

- Downsize first

  If you are going to interpolate in time, depth, and lon/lat, consider if one of those interpolation steps will result in much less model output, and if so, do that step first. For example, if you will interpolate to 3 data locations in lon/lat but 50 vertical levels, first interpolate in lon/lat before interpolating in z to save time.

- Chunking

  A DataArray cannot be chunked in the dimension that is being interpolated on. So, in the previous example of interpolating first in lon/lat, the DataArray can have dask chunks in the Z and T directions when calculating the lon/lat interpolation. Then, the DataArray would need to be rechunked so that no chunks are in the Z dimension before interpolating in the Z dimension. Similarly for time. You can check chunks with `da.chunks`, specify new chunks with `da.chunk({'ocean_time': 1, 's_rho': 5})` and reset any individual dimension chunking by passing in -1, or reset all chunks for a DataArray or Dataset with `ds.chunk(-1)`.

```
varin = ds.salt
lons, lats = [-93, -92, -91], [28, 28.5, 29]
zs = np.linspace(0, -50, 20)
startdate = pd.Timestamp(ds.ocean_time[0].values)
ts = [startdate + pd.Timedelta('30 min')*i for i in range(10)]
ts = xr.DataArray(ts, dims='ocean_time', attrs={'axis': 'T', 'standard_name': 'time'})
```

Since there are only a few lons/lats, I will start with that:

```
varout = xroms.interpll(varin, lons, lats, which='pairs')
print(varout)
```

The order of the other two steps probably doesn't matter too much in this case:

```
varout
```

```
varout2 = varout.interp(ocean_time=ts)
varout3 = xroms.isoslice(varout2, zs, xgrid)
# print(varout3)
```

Note that `cf-xarray` still works on this output:

```
varout3.cf.describe()
```

Cross-section or isoslice

A cross-section or isoslice can be calculated using `xroms.isoslice`. A short example is given here, but more examples are given in the `xroms.isoslice` docs. This is the same function used for interpolating variables to fixed depths as demonstrated earlier in this notebook.

Calculate cross-section of u-velocity along latitude of 27 degrees.

```
grid = ds.xroms.xgrid
lat0 = 27
varin = ds.u
```

```
xroms.isoslice(varin, np.array([lat0]), xgrid, iso_array=varin.cf['latitude'], axis='Y')
```

```
import xarray as xr
import xroms
import matplotlib.pyplot as plt
import cartopy
import numpy as np
# import hvplot.xarray
# import geoviews as gv
import cmocean.cm as cmo
import xcmocean
```

1.5 How to plot

This notebook demonstrates how to plot ROMS model output from a planview (x-y) and an x-z cross-section. Static and interactive approaches are shown. The `cartopy` package is used for managing projections for mapview plots, which also gives many options for input (some shown below).

Note you need version 0.11 of Datashader for rasterizing to work in the interactive plots. (https://github.com/holoviz/hvplot/issues/434)

1.5.1 Load in data

Load in example dataset. More information at in *input/output page*

```
ds = xroms.datasets.fetch_ROMS_example_full_grid()
ds, xgrid = xroms.roms_dataset(ds, include_cell_volume=True, include_Z0=True)
ds.xroms.set_grid(xgrid)
ds
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
```

(continues on next page)

(continued from previous page)

```
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
```

(continues on next page)

(continued from previous page)

```
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

(continues on next page)

(continued from previous page)

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
```

(continues on next page)

(continued from previous page)

```
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
```

(continues on next page)

(continued from previous page)

```
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
```

(continues on next page)

(continued from previous page)

```
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
```

(continues on next page)

(continued from previous page)

```
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/xgcm/grid_ufunc.py:832: FutureWarning: The return type of `Dataset.dims` will␣
↪be changed to return a set of dimension names in future, in order to be more␣
↪consistent with `DataArray.dims`. To access a mapping from dimension names to lengths,␣
↪please use `Dataset.sizes`.
  out_dim: grid._ds.dims[out_dim] for arg in out_core_dims for out_dim in arg
```

```
<xarray.Dataset> Size: 957MB
Dimensions:     (eta_rho: 191, xi_rho: 300, s_rho: 30, s_w: 31, ocean_time: 2,
                 xi_u: 299, eta_v: 190)
Coordinates: (12/29)
    lon_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    lat_rho     (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
  * s_rho       (s_rho) float64 240B -0.9833 -0.95 -0.9167 ... -0.05 -0.01667
  * s_w         (s_w) float64 248B -1.0 -0.9667 -0.9333 ... -0.03333 0.0
  * ocean_time  (ocean_time) datetime64[ns] 16B 2009-11-19T12:00:00 2009-11-1...
    lon_u       (eta_rho, xi_u) float64 457kB dask.array<chunksize=(191, 299), meta=np.
↪ndarray>
    ...          ...
    z_rho_v0    (s_rho, eta_v, xi_rho) float64 14MB dask.array<chunksize=(30, 190, 300),␣
```

(continues on next page)

(continued from previous page)

```
↪meta=np.ndarray>
    z_rho_psi0  (s_rho, eta_v, xi_u) float64 14MB dask.array<chunksize=(30, 190, 299),␣
↪meta=np.ndarray>
    z_w0        (s_w, eta_rho, xi_rho) float64 14MB dask.array<chunksize=(31, 191, 300),␣
↪meta=np.ndarray>
    z_w_u0      (s_w, eta_rho, xi_u) float64 14MB dask.array<chunksize=(31, 191, 299),␣
↪meta=np.ndarray>
    z_w_v0      (s_w, eta_v, xi_rho) float64 14MB dask.array<chunksize=(31, 190, 300),␣
↪meta=np.ndarray>
    z_w_psi0    (s_w, eta_v, xi_u) float64 14MB dask.array<chunksize=(31, 190, 299),␣
↪meta=np.ndarray>
Data variables: (12/53)
    angle       (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    hc          float64 8B ...
    Cs_r        (s_rho) float64 240B dask.array<chunksize=(30,), meta=np.ndarray>
    zeta        (ocean_time, eta_rho, xi_rho) float32 458kB dask.array<chunksize=(2, 191,
↪ 300), meta=np.ndarray>
    h           (eta_rho, xi_rho) float64 458kB dask.array<chunksize=(191, 300), meta=np.
↪ndarray>
    Cs_w        (s_w) float64 248B dask.array<chunksize=(31,), meta=np.ndarray>
    ...          ...
    dV_w_u      (ocean_time, s_w, eta_rho, xi_u) float64 28MB dask.array<chunksize=(2,␣
↪31, 191, 299), meta=np.ndarray>
    dV_v        (ocean_time, s_rho, eta_v, xi_rho) float64 27MB dask.array<chunksize=(2,␣
↪30, 190, 300), meta=np.ndarray>
    dV_w_v      (ocean_time, s_w, eta_v, xi_rho) float64 28MB dask.array<chunksize=(2,␣
↪31, 190, 300), meta=np.ndarray>
    dV_psi      (ocean_time, s_rho, eta_v, xi_u) float64 27MB dask.array<chunksize=(2,␣
↪30, 190, 299), meta=np.ndarray>
    dV_w_psi    (ocean_time, s_w, eta_v, xi_u) float64 28MB dask.array<chunksize=(2, 31,␣
↪190, 299), meta=np.ndarray>
    rho0        int64 8B 1025
Attributes: (12/29)
    file:               ocean_his_0150.nc
    format:             netCDF-3 classic file
    Conventions:        CF-1.4
    type:               ROMS/TOMS history file
    title:              Texas and Louisiana Shelf case (Nesting)
    rst_file:           ocean_rst.nc
    ...                 ...
    compiler_command:   /g/software/openmpi/1.4.3/intel/bin/mpif90
    compiler_flags:     -heap-arrays -fp-model precise -assume 2underscores -c...
    tiling:             016x032
    history:            ROMS/TOMS, Version 3.4, Sunday - December 4, 2011 -  7...
    ana_file:           /scratch/zhangxq/projects/txla_nesting6/Functionals/an...
    CPP_options:        TXLA, ANA_BSFLUX, ANA_BTFLUX, ASSUMED_SHAPE, BULK_FLUX...
```

1.5.2 Setup for plotting

Use `cartopy` when plotting with a projection and/or wanting to add context like coastline.

```
proj = cartopy.crs.LambertConformal(central_longitude=-98, central_latitude=30)
pc = cartopy.crs.PlateCarree()
```

1.5.3 Using `cf-xarray` with plots

As described in the *select data page*, xroms is built to use the cf-xarray accessor to make working with dimensions easier.

See description of a DataArray with

```
ds.salt.cf.describe()
```

which returns

```
ds.salt.cf.describe()
```

```
Coordinates:
            CF Axes: * X: ['xi_rho']
                     * Y: ['eta_rho']
                     * Z: ['s_rho']
                     * T: ['ocean_time']

     CF Coordinates:   longitude: ['lon_rho']
                       latitude: ['lat_rho']
                       vertical: ['z_rho']
                     * time: ['ocean_time']

      Cell Measures:   area, volume: n/a

     Standard Names:   depth: ['z_rho']
                       latitude: ['lat_rho']
                       longitude: ['lon_rho']
                     * ocean_s_coordinate_g1: ['s_rho']
                     * time: ['ocean_time']

             Bounds:   n/a

      Grid Mappings:   n/a
```

```
/tmp/ipykernel_2909/990108105.py:1: DeprecationWarning: 'obj.cf.describe()' will be␣
↪removed in a future version. Use instead 'repr(obj.cf)' or 'obj.cf' in a Jupyter␣
↪environment.
  ds.salt.cf.describe()
```

From this you know that you can use the `cf-xarray` accessor to call many typical `xarray` functions from your Dataset or DataArray by substituting the generic Axes names (X, Y, Z, or T) in place of the specific dimension name on the right side of the Axes description. For example:

```
ds.salt.cf.sel(X=20, T='2010-1-1')
```

instead of

```
ds.salt.sel(xi_rho=20, ocean_time='2010-1-1')
```

A similar syntax works for `isel` commands.

You also know from this description that calls that take in coordinates for `xarray` can be used with their generic coordinate name listed above (longitude, latitude, vertical, time). For example:

```
ds.salt.cf.plot(x='longitude', y='latitude')
```

instead of

```
ds.salt.plot(x='lon_rho', y='lat_rho')
```

We use these shorter generalized Axes and Coordinates below.

1.5.4 `xcmocean` for choosing colormaps

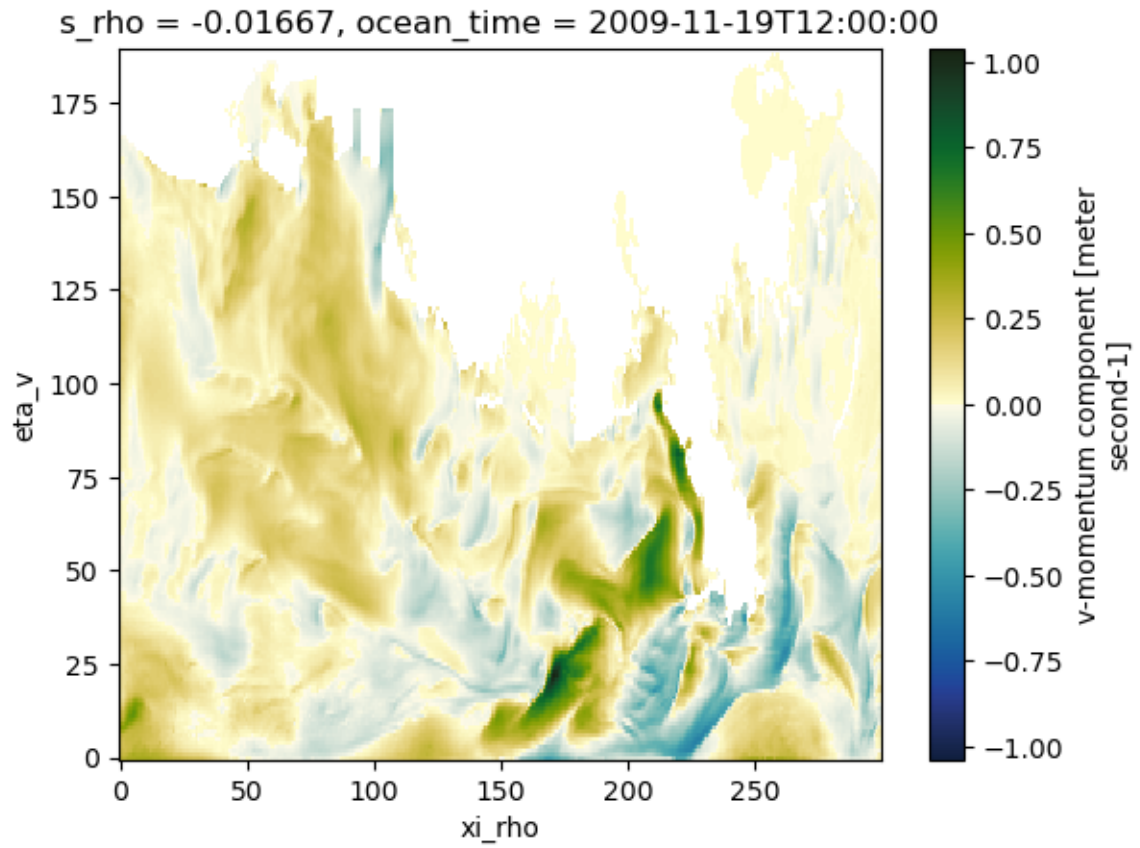
The `xcmocean` accessor can be used when plotting with `xarray` to automatically choose a "good" colormap for the plot based on the DataArray name and attributes and colormaps from the `cmocean` colormaps package.

To use the colormap accessor, add `.cmo` before the plot call from xarray (but skip the `plot` if specifying the type of plot subsequently, like `pcolormesh`). These are the call options:

- `cmo.plot()`
- `cmo.pcolormesh()` to specify `pcolormesh`
- `cmo.contourf()` to specify `contourf`
- `cmo.contour()` to specify `contour`

```
ds.v.cf.isel(T=0, Z=-1).cmo.pcolormesh()
```

```
<matplotlib.collections.QuadMesh at 0x7f50c96698d0>
```

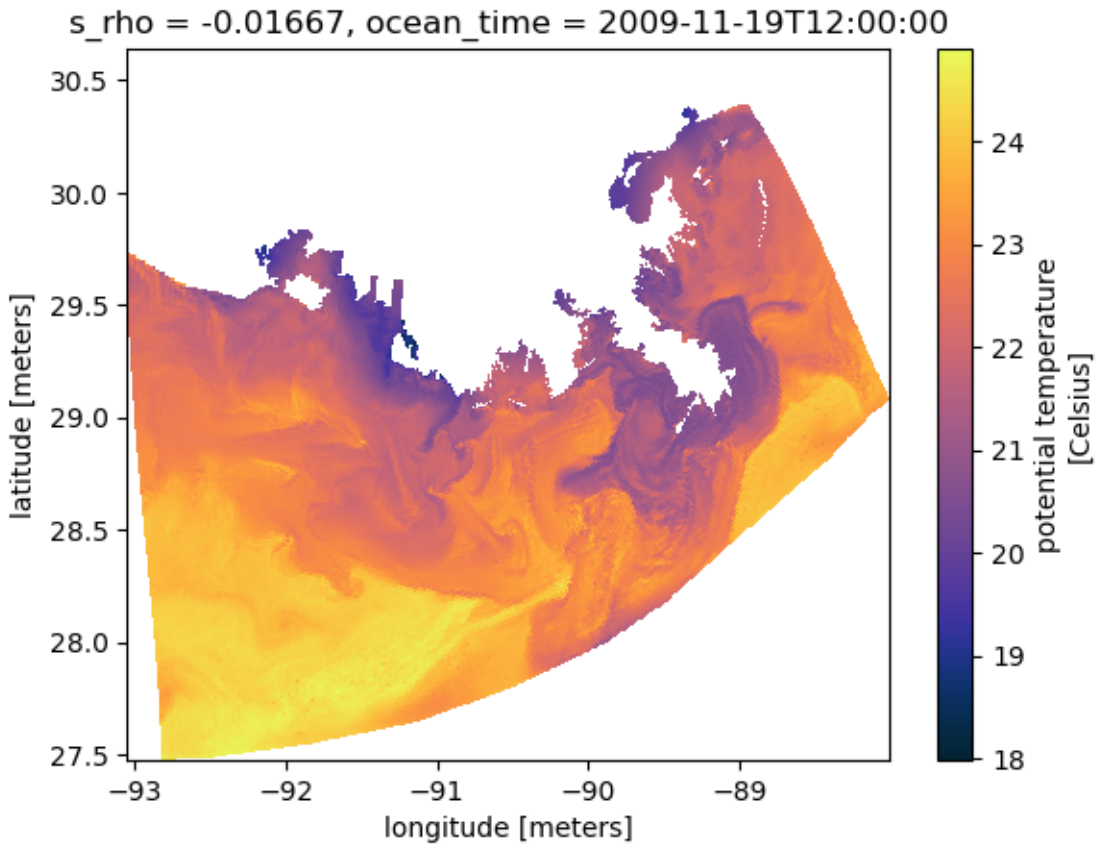


xcmocean can also be used in conjunction with cf-xarray but with slightly different syntax. You can access the same plot options while using cf-xarray in the plot call with the following:

- cmo.cfplot()
- cmo.cfpcolormesh()
- cmo.cfcontourf()
- cmo.cfcontour()

```
ds.temp.cf.isel(T=0, Z=-1).cmo.cfpcolormesh(x='longitude', y='latitude')
```

```
<matplotlib.collections.QuadMesh at 0x7f50be2a6e60>
```



1.5.5 Static: `xarray`

Can plot directly in `xarray` since it has wrapped many `Matplotlib` plotting routines. This is great for quick plots and continues to improve, but if you need more control then try the next section of plotting directly in `Matplotlib`.

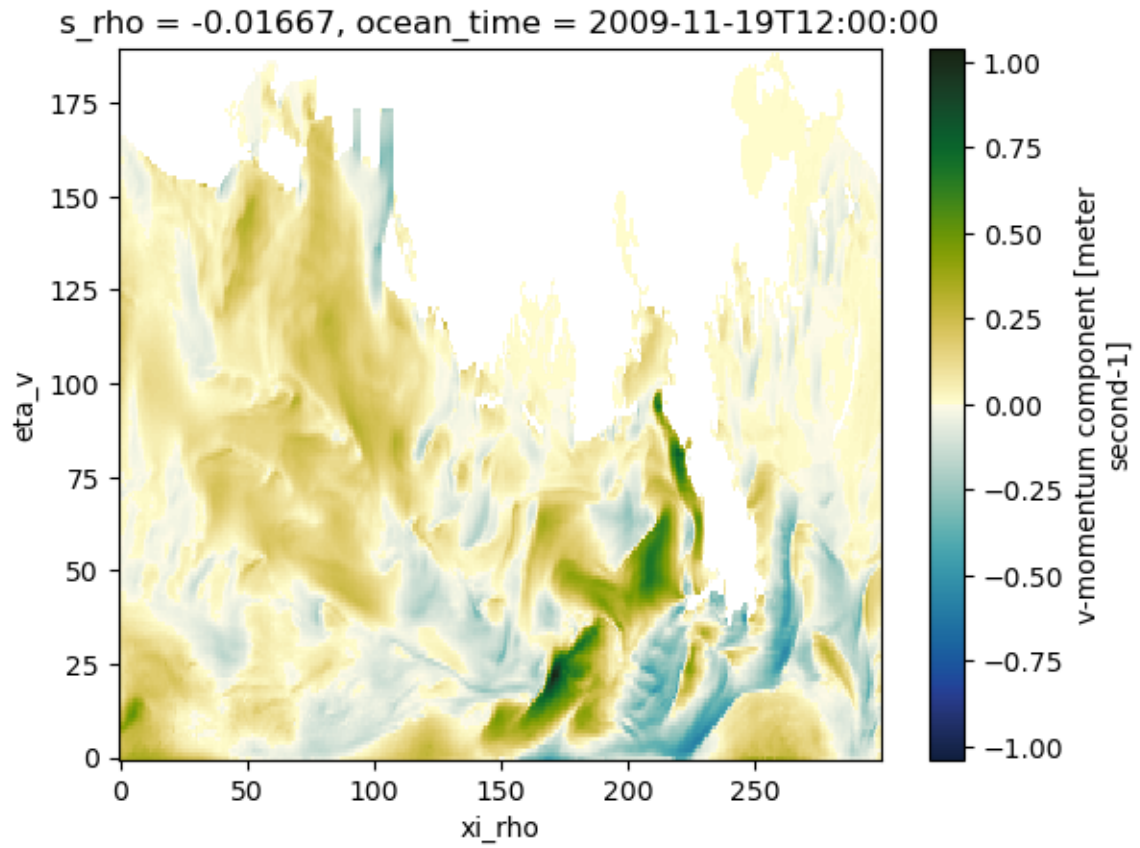
map view

Overview

Plotted with dimension indices instead of coordinates:

```
ds.v.cf.isel(T=0, Z=-1).plot(cmap=cmo.delta)
```

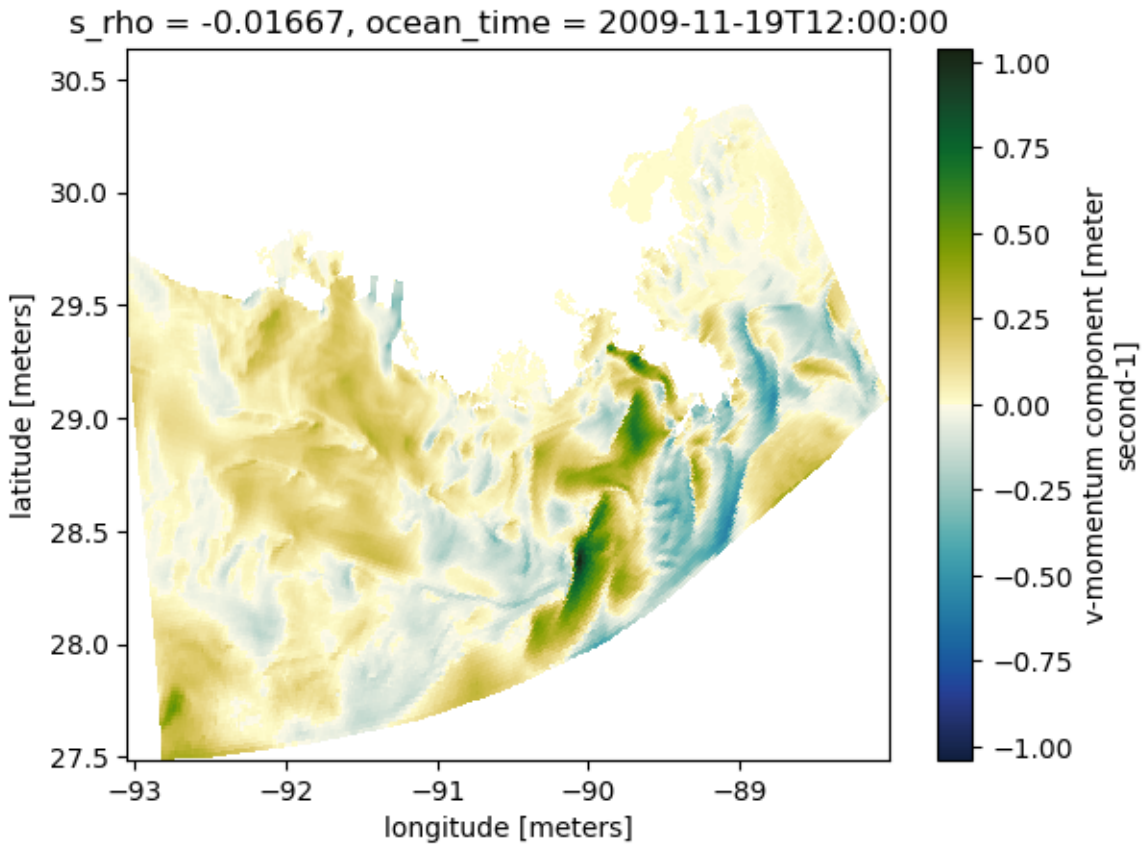
```
<matplotlib.collections.QuadMesh at 0x7f50bdfbec50>
```



Plotted with coordinates lon/lat:

```
ds.v.cf.isel(T=0, Z=-1).cf.plot(x='longitude', y='latitude', cmap=cmo.delta)
```

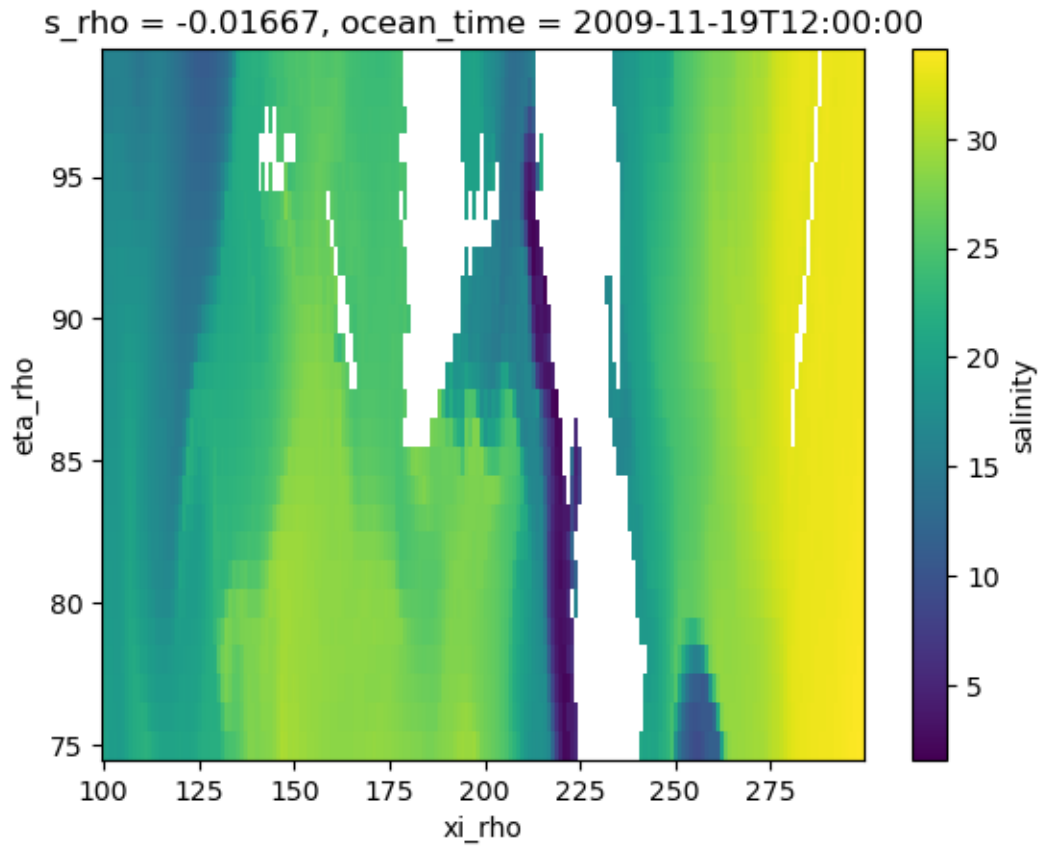
```
<matplotlib.collections.QuadMesh at 0x7f50be0a19f0>
```



magnified

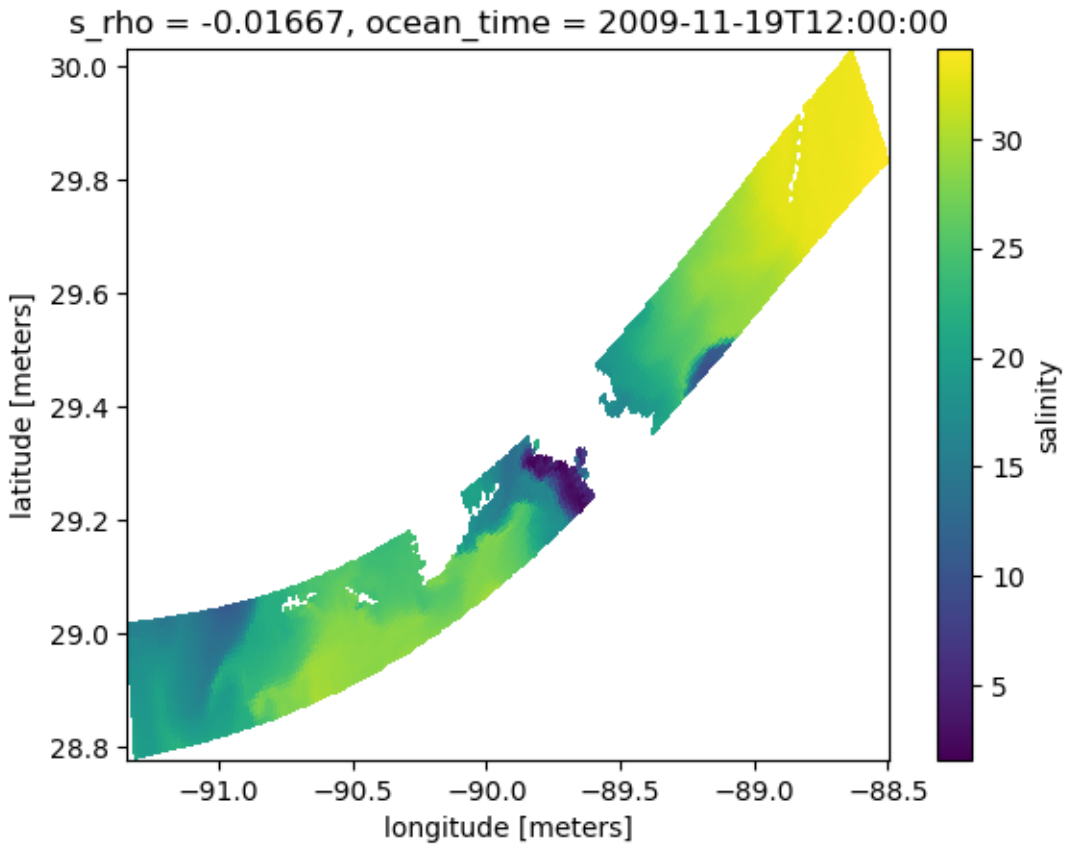
```
ds.salt.cf.isel(T=0, Z=-1, X=slice(100,300), Y=slice(75,100)).plot()
```

```
<matplotlib.collections.QuadMesh at 0x7f50bd7934c0>
```




```
ds.salt.cf.isel(T=0, Z=-1, X=slice(100,300), Y=slice(75,100)).cf.plot(x='longitude', y=
↪'latitude')
```

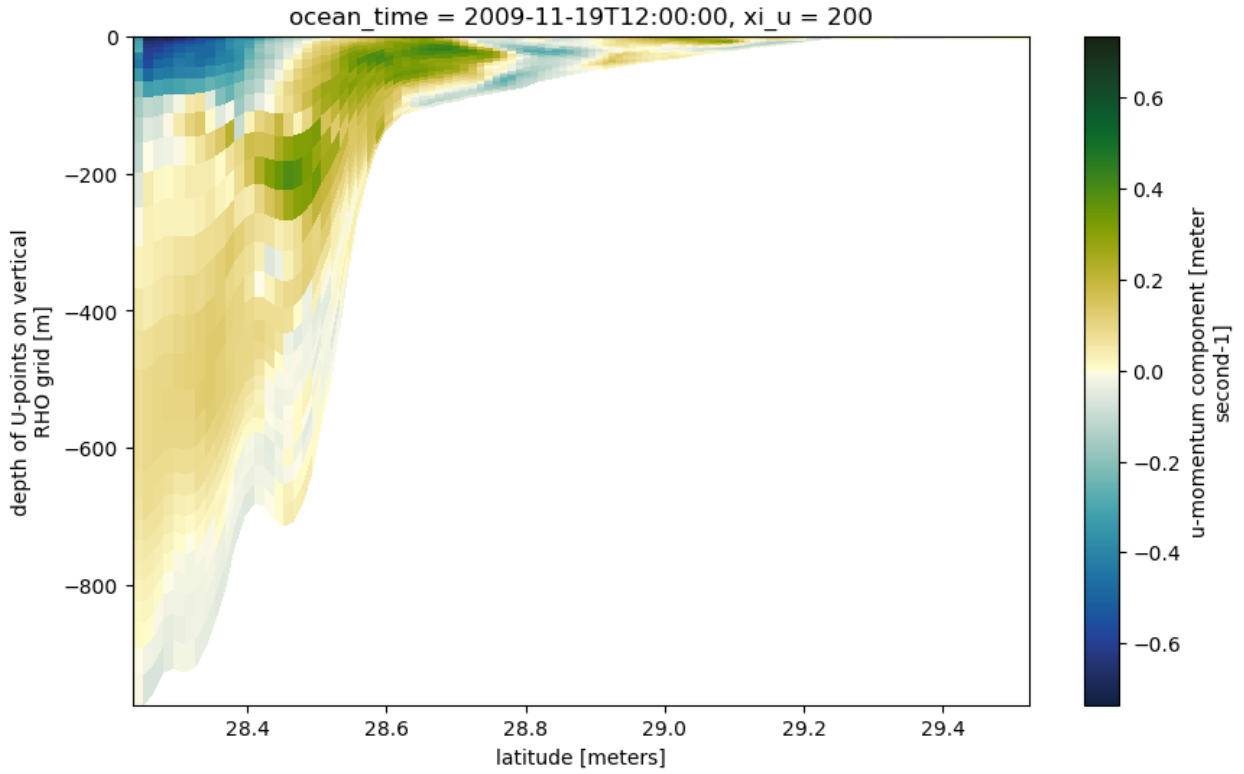
```
<matplotlib.collections.QuadMesh at 0x7f50bd6577f0>
```



cross-section

```
dss = ds.u.cf.isel(X=200, T=0)
dss.where(~dss.isnull().compute(), drop=True).cf.plot(x='latitude', y='vertical',␣
↪cmap=cmo.delta, figsize=(10,6))
```

```
<matplotlib.collections.QuadMesh at 0x7f50bd538cd0>
```



1.5.6 Static: `Matplotlib`

map view

Overview

Here is a basic plan-view map, using `cartopy` for projection handling. You can add many different types of natural data for context. Shown here are coastline, land, rivers, and state borders. You can control the resolution of the data by changing the input to `with_scale` (options are '10m', '50m', or '110m', corresponding to 1:10,000,000, 1:50,000,000, and 1:110,000,000 scale). More `cartopy` feature information available here: https://scitools.org.uk/cartopy/docs/v0.16/matplotlib/feature_interface.html.

```
fig = plt.figure(figsize=(12,8))
ax = plt.axes(projection=proj)

# Add natural features
ax.add_feature(cartopy.feature.LAND.with_scale('110m'), facecolor='0.8')
ax.add_feature(cartopy.feature.COASTLINE.with_scale('10m'), edgecolor='0.2')
ax.add_feature(cartopy.feature.RIVERS.with_scale('110m'), edgecolor='b')
ax.add_feature(cartopy.feature.STATES.with_scale('110m'), edgecolor='k')

gl = ax.gridlines(draw_labels=True, x_inline=False, y_inline=False, xlocs=np.arange(-104,
↪-80,2))

# manipulate `gridliner` object to change locations of labels
gl.top_labels = False
```

(continues on next page)

(continued from previous page)

```
gl.right_labels = False

ds.salt.cf.isel(T=0, Z=-1).cf.plot(ax=ax, x='longitude', y='latitude', transform=pc)
```

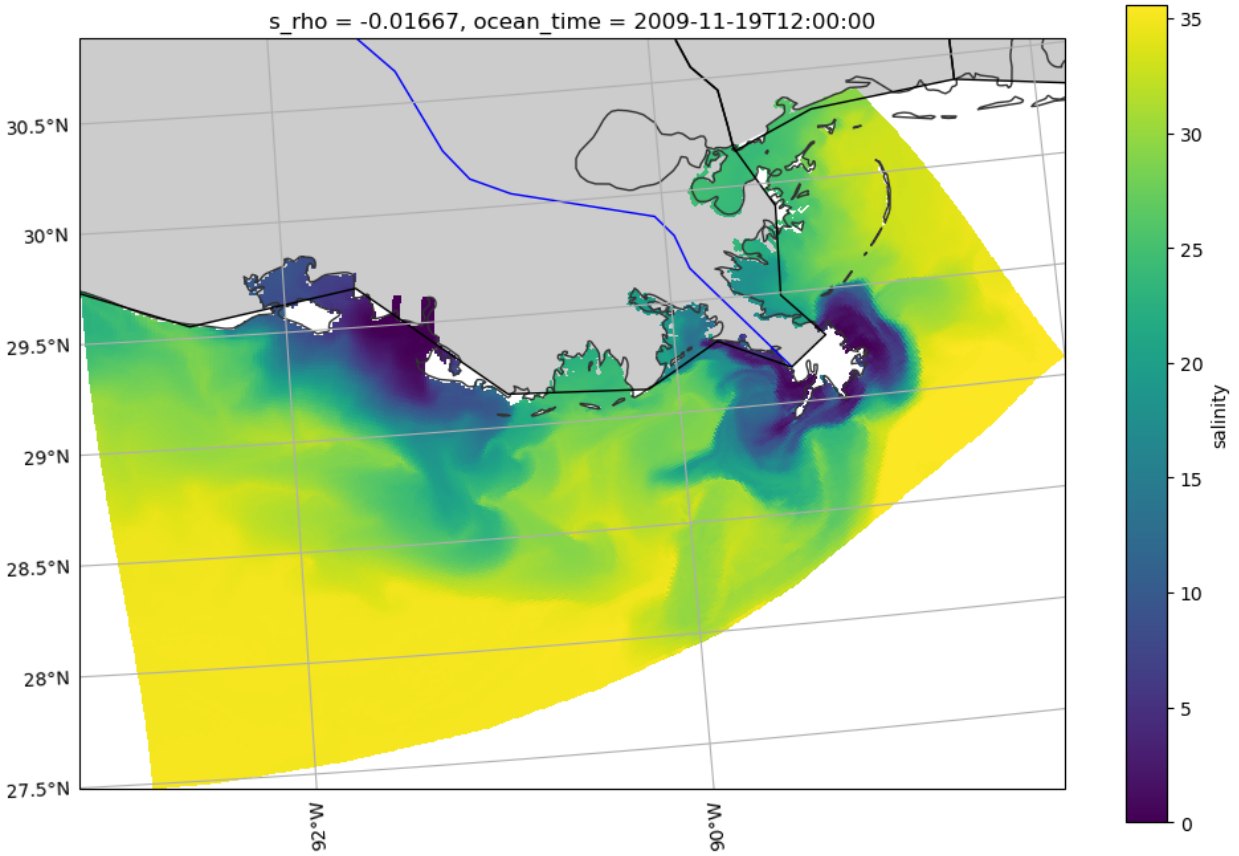
```
<cartopy.mpl.geocollection.GeoQuadMesh at 0x7f50bd45f220>
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/cartopy/io/__init__.py:241: DownloadWarning: Downloading: https://
↪naturalearth.s3.amazonaws.com/110m_physical/ne_110m_land.zip
  warnings.warn(f'Downloading: {url}', DownloadWarning)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/cartopy/io/__init__.py:241: DownloadWarning: Downloading: https://
↪naturalearth.s3.amazonaws.com/10m_physical/ne_10m_coastline.zip
  warnings.warn(f'Downloading: {url}', DownloadWarning)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/cartopy/io/__init__.py:241: DownloadWarning: Downloading: https://
↪naturalearth.s3.amazonaws.com/110m_physical/ne_110m_rivers_lake_centerlines.zip
  warnings.warn(f'Downloading: {url}', DownloadWarning)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/xroms/conda/latest/lib/python3.10/site-
↪packages/cartopy/io/__init__.py:241: DownloadWarning: Downloading: https://
↪naturalearth.s3.amazonaws.com/110m_cultural/ne_110m_admin_1_states_provinces_lakes.zip
  warnings.warn(f'Downloading: {url}', DownloadWarning)
```

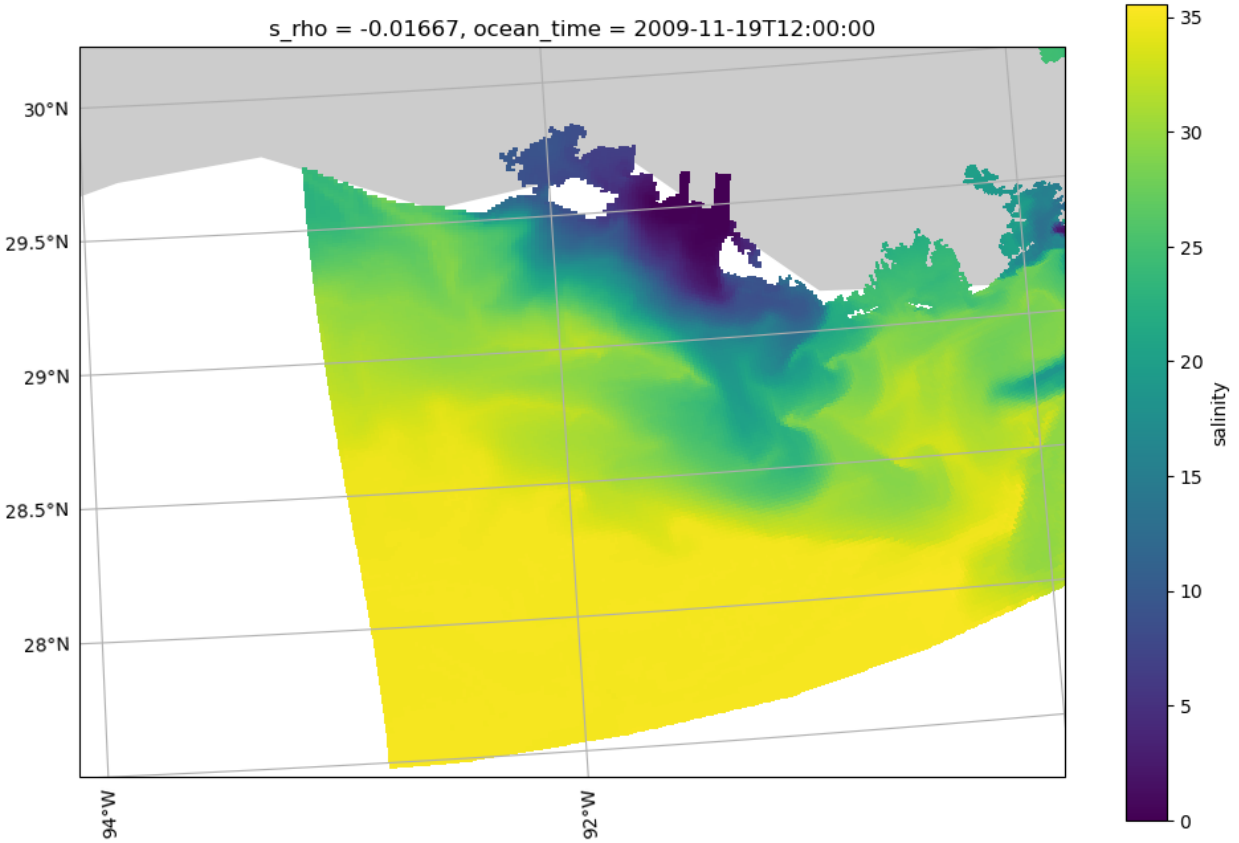


Magnified

Use `set_extent` to narrow the view to magnify a subregion.

```
fig = plt.figure(figsize=(12,8))
ax = plt.axes(projection=proj)

ax.set_extent([-94, -90, 27.5, 30], crs=pc)
ax.add_feature(cartopy.feature.LAND.with_scale('110m'), facecolor='0.8')
gl = ax.gridlines(draw_labels=True, x_inline=False, y_inline=False, xlocs=np.arange(-104,
↪-80,2))

# manipulate `gridliner` object to change locations of labels
gl.top_labels = False
gl.right_labels = False

ds.salt.cf.isel(T=0, Z=-1).cf.plot(ax=ax, x='longitude', y='latitude', transform=pc)
```

```
<cartopy.mpl.geocollection.GeoQuadMesh at 0x7f50bd755f60>
```

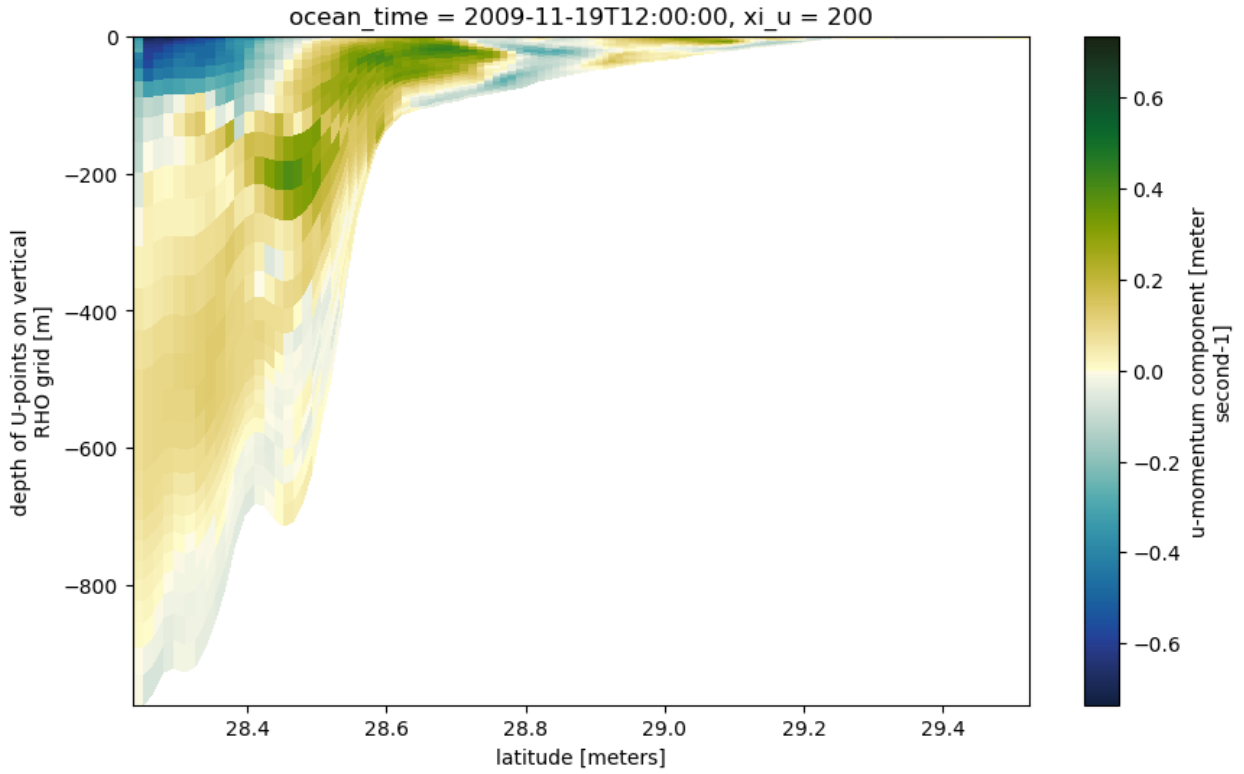


cross-section

This is the same as the example above for plotting directly from `xarray` since projections on plan-view maps make most of the difference.

```
dss = ds.u.cf.isel(X=200, T=0)
dss.where(~dss.isnull().compute(), drop=True).cf.plot(x='latitude', y='vertical',␣
↪cmap=cmo.delta, figsize=(10,6))
```

```
<matplotlib.collections.QuadMesh at 0x7f50bd1f9240>
```



1.5.7 Interactive

In these interactive plots, you can zoom, pan, and save plots using the menu on the right-hand side. There is a mouse hover option to display specific values; this can also be turned off. The plot automatically makes a widget to the right of the plot to easily vary over that variable. Currently the plots below are set to vary over time.

These plots aren't working interactively in the docs, but are left here as examples for your own use.

map view

The tiles allow for different basemap options. The `rasterize` option is really important here by allowing a lower resolution presentation when zoomed out and increasing resolution with magnification, potentially saving a lot of time.

Vary over time (for surface)

```
tiles = gv.tile_sources.ESRI  # optional, for a basemap
ds.salt.cf.isel(s_rho=-1).hvplot.quadmesh(x='lon_rho', y='lat_rho', width=650,␣
↪height=500,
                        cmap="cmo.haline", rasterize=True, crs=pc) * tiles
```

Vary over sigma level and time

```
tiles = gv.tile_sources.ESRI  # optional, for a basemap
ds.salt.hvplot.quadmesh(x='lon_rho', y='lat_rho', width=650, height=500,
                        cmap=cmo.haline, rasterize=True, crs=pc) * tiles
```

Vary over depth and time

Since the vertical dimension is in sigma coordinates instead of fixed depths, it is not immediate to be able to use a widget to vary depth in these plots. However, it is still possible to do using `xroms` code. First set up the calculation for several depths you want to examine using `xroms.isoslice`, then send that `xarray` object to `hvplot` for plotting. It is slow because it has to calculate everything, but it is nevertheless interactive. Having these files locally would speed it up.

In the following example, we use the accessor version of the isoslice interpolation to find slices of salinity at fixed depths. To save some time, we use the time-constant depths (`z_rho0`) associated with salinity instead of the time-varying version (`z_rho`).

```
zsalt = ds.salt.xroms.isoslice([-10, -20, -30], iso_array=ds.salt.z_rho0, axis='Z')
```

```
tiles = gv.tile_sources.ESRI  # optional, for a basemap
zsalt.hvplot.quadmesh(x='lon_rho', y='lat_rho', width=650, height=500,
                        cmap=cmo.haline, rasterize=True, crs=pc) * tiles
```

cross-section

In this case, the plots are similar whether `rasterize=True` is used or not.

```
ds.temp.isel(xi_rho=300).hvplot.quadmesh(x='lat_rho', y='z_rho0', width=750, height=400,
                        cmap=cmo.thermal)
```

1.6 API

xroms	Functions to help read in ROMS output.
derived	Variables derived from ROMS output are here.
interp	Interpolation functions.
roms_seawater	Functions related to density of seawater.
utilities	Functions that act on DataArrays or Datasets.
vector	Functions related to vectors.
accessor	This is an accessor to xarray.

1.6.1 xroms.xroms

Functions to help read in ROMS output.

xroms.xroms.**open_mfnetcdf**(*files*, *chunks={'ocean_time': 1}*, *xrargs={}*, *Vtransform=None*, *add_verts=False*, *proj=None*)

Return Dataset based on a list of netCDF files.

This function is deprecated; use *xroms.open_netcdf* or *xroms.open_zarr* directly instead.

Parameters

- **files** (*list of strings*) – Where to find the model output. *files* can be a list of netCDF file names.
- **chunks** (*dict, optional*) – The specified chunks for the Dataset. Use chunks to read in output using dask.
- **xrargs** (*dict, optional*) – Keyword arguments to be passed to *xarray.open_mfdataset*. Anything input by the user overwrites the default selections saved in this function. Defaults are:

  {'compat': 'override', 'combine': 'by_coords',
  'data_vars': 'minimal', 'coords': 'minimal', 'parallel': True}

  Many other options are available; see xarray docs.
- **Vtransform** (*int, optional*) – Vertical transform for ROMS model. Should be either 1 or 2 and only needs to be input if not available in ds.
- **add_verts** (*boolean, optional*) – Add 'verts' horizontal grid to ds if True. This requires a cartopy projection to be input too. This is passed to *roms_dataset*.
- **proj** (*cartopy crs projection, optional*) – Should match geographic area of model domain. Required if *add_verts=True*, otherwise not used. This is passed to *roms_dataset*. Example: >>> proj = cartopy.crs.LambertConformal(central_longitude=-98, central_latitude=30)

Returns

ds – Model output, read into an *xarray* Dataset. If 'chunks' keyword argument is input, dask is used when reading in model output and output is read in lazily instead of eagerly.

Return type

Dataset

Examples

```
>>> ds = xroms.open_mfnetcdf(files)
```

xroms.xroms.**open_netcdf**(*file*, *chunks={'ocean_time': 1}*, *xrargs={}*, *Vtransform=None*, *add_verts=False*, *proj=None*)

Return Dataset based on a single thredds or physical location.

This function is deprecated; use *xroms.open_netcdf* or *xroms.open_zarr* directly instead.

Parameters

- **file** (*str*) – Where to find the model output. *file* could be: * a string of a single netCDF file name, or * a string of a thredds server address containing model output.

- **chunks** (*dict, optional*) – The specified chunks for the Dataset. Use chunks to read in output using dask.
- **xrargs** (*dict, optional*) – Keyword arguments to be passed to *xarray.open_dataset*. See *xarray* docs for options.
- **Vtransform** (*int, optional*) – Vertical transform for ROMS model. Should be either 1 or 2 and only needs to be input if not available in ds.
- **add_verts** (*boolean, optional*) – Add 'verts' horizontal grid to ds if True. This requires a cartopy projection to be input too. This is passed to *roms_dataset*.
- **proj** (*cartopy crs projection, optional*) – Should match geographic area of model domain. Required if *add_verts=True*, otherwise not used. This is passed to *roms_dataset*. Example: >>> proj = cartopy.crs.LambertConformal(central_longitude=-98, central_latitude=30)

Returns

ds – Model output, read into an *xarray* Dataset. If 'chunks' keyword argument is input, dask is used when reading in model output and output is read in lazily instead of eagerly.

Return type

Dataset

Examples

```
>>> ds = xroms.open_netcdf(file)
```

xroms.xroms.**open_zarr**(*files*, *chunks={'ocean_time': 1}*, *xrargs={}*, *xrconcatargs={}*, *Vtransform=None*, *add_verts=False*, *proj=None*)

Return a Dataset based on a list of zarr files

Parameters

- **files** (*list of strings*) – A list of zarr file directories.
- **chunks** (*dict, optional*) – The specified chunks for the Dataset. Use chunks to read in output using dask.
- **xrargs** (*dict, optional*) – Keyword arguments to be passed to *xarray.open_zarr*. Anything input by the user overwrites the default selections saved in this function. Defaults are:

  {'consolidated': True, 'drop_variables': 'dstart'}

  Many other options are available; see xarray docs.
- **xrconcatargs** (*dict, optional*) – Keyword arguments to be passed to *xarray.concat* for combining zarr files together. Anything input by the user overwrites the default selections saved in this function. Defaults are:

  {'dim': 'ocean_time', 'data_vars': 'minimal', 'coords': 'minimal'}

  Many other options are available; see xarray docs.
- **Vtransform** (*int, optional*) – Vertical transform for ROMS model. Should be either 1 or 2 and only needs to be input if not available in ds.
- **add_verts** (*boolean, optional*) – Add 'verts' horizontal grid to ds if True. This requires a cartopy projection to be input too. This is passed to *roms_dataset*.

- **proj** (*cartopy crs projection, optional*) – Should match geographic area of model domain. Required if *add_verts=True*, otherwise not used. This is passed to *roms_dataset*. Example: >>> proj = cartopy.crs.LambertConformal(central_longitude=-98, central_latitude=30)

Returns

ds – Model output, read into an *xarray* Dataset. If 'chunks' keyword argument is input, dask is used when reading in model output and output is read in lazily instead of eagerly.

Return type

Dataset

Examples

```
>>> ds = xroms.open_zarr(files)
```

xroms.xroms.**roms_dataset**(*ds*, *Vtransform=None*, *add_verts=False*, *proj=None*, *include_Z0=False*, *include_3D_metrics=True*, *include_cell_volume=False*, *include_cell_area=False*)

Modify Dataset to be aware of ROMS coordinates, with matching xgcm grid object.

Parameters

- **ds** (*Dataset*) – xarray Dataset with model output
- **Vtransform** (*int, optional*) – Vertical transform for ROMS model. Should be either 1 or 2 and only needs to be input if not available in ds.
- **add_verts** (*boolean, optional*) – Add 'verts' horizontal grid to ds if True. This requires a cartopy projection to be input too.
- **proj** (*cartopy crs projection, optional*) – Should match geographic area of model domain. Required if *add_verts=True*, otherwise not used. Example: >>> proj = cartopy.crs.LambertConformal(central_longitude=-98, central_latitude=30)
- **include_Z0** (*bool*) – If True, calculate depths for quiescient state, which can be used for faster approximations for depth calculations since they are 3D instead of 4D.
- **include_3D_metrics** (*bool*) – If True, calculate necessary grid metrics for 3D calculations with xgcm. Note that you need the 3D metrics for horizontal derivatives for ROMS.
- **include_cell_volume** (*bool*) – If True, calculate necessary grid metrics for cell volumes. I think this is for cf-xarray.
- **include_cell_area** (*bool*) – If True, calculate necessary grid metrics for cell areas (besides dA). I think this is for cf-xarray.

Returns

- **ds** (*Dataset*) – Same dataset as input, but with dimensions renamed to be consistent with *xgcm* and with vertical coordinates and metrics added.
- **xgrid** (*xgcm grid object*) – Includes ROMS metrics so can be used for xgcm grid operations, which mostly have been wrapped into xroms.

Notes

Note that this could be very slow if dask is not on.

This does not need to be run by the user if *xroms* functions *open_netcdf* or *open_zarr* are used for reading in model output, since run in those functions.

This also uses *cf-xarray* to manage dimensions of variables.

Examples

```
>>> ds, xgrid = xroms.roms_dataset(ds)
```

Modules

1.6.2 xroms.derived

Variables derived from ROMS output are here.

Functions

EKE(ug, vg, xgrid[, hboundary, hfill_value])	Calculate EKE [m^2/s^2]
KE(rho0, speed)	Calculate kinetic energy [kg/(m*s^2)]
convergence(u, v, xgrid[, hboundary, ...])	Calculate 2D convergence from u and v [1/s].
dudz(u, xgrid[, sboundary, sfill_value])	Calculate the xi component of vertical shear [1/s]
dvdz(v, xgrid[, sboundary, sfill_value])	Calculate the eta component of vertical shear [1/s]
ertel(phi, u, v, f, xgrid[, hcoord, scoord, ...])	Calculate Ertel potential vorticity of phi.
omega(u, v)	Calculate s-grid vertical velocity from u and v [m/s]
relative_vorticity(u, v, xgrid[, hboundary, ...])	Calculate the vertical component of the relative vorticity [1/s]
speed(u, v, xgrid[, hboundary, hfill_value])	Calculate horizontal speed [m/s] from u and v components
uv_geostrophic(zeta, f, xgrid[, hboundary, ...])	Calculate geostrophic velocities from zeta [m/s]
vertical_shear(dudz, dvdz, xgrid[, ...])	Calculate the vertical shear [1/s]
w(u, v)	Calculate vertical velocity from u and v [m/s]

xroms.derived.**EKE**(*ug*, *vg*, *xgrid*, *hboundary='extend'*, *hfill_value=None*)

Calculate EKE [m^2/s^2]

Parameters

- **ug** (*DataArray*) – Geostrophic or other xi component velocity [m/s]
- **vg** (*DataArray*) – Geostrophic or other eta component velocity [m/s]
- **xgrid** (*xgcm.grid*) – Grid object associated with ug, vg
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for moving to rho grid. From xgcm documentation: A flag indicating how to handle boundaries:

- None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
- 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
- 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection for moving to rho grid. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

Returns

- *DataArray of eddy kinetic energy on rho grid.*
- Output is *[T,Y,X]*.

Notes

EKE = 0.5*(ug^2 + vg^2)

Examples

```
>>> ug, vg = xroms.uv_geostrophic(ds.zeta, ds.f, xgrid)
>>> xroms.EKE(ug, vg, xgrid)
```

xroms.derived.**KE**(*rho0*, *speed*)

Calculate kinetic energy [kg/(m*s^2)]

Parameters

- **rho0** (*float*) – background density of the water [kg/m^3]
- **speed** (*DataArray*) – magnitude of horizontal velocity vector [m/s]

Returns

- *DataArray of kinetic energy on rho/rho grids.*
- Output is *[T,Z,Y,X]*.

Notes

KE = 0.5*rho*(u^2 + v^2)

Examples

```
>>> speed = xroms.speed(ds.u, ds.v, xgrid)
>>> xroms.KE(ds.rho0, speed)
```

xroms.derived.**convergence**(*u*, *v*, *xgrid*, *hboundary='extend'*, *hfill_value=None*, *sboundary='extend'*, *sfill_value=None*)

Calculate 2D convergence from u and v [1/s].

Parameters

- **u** (*DataArray*) – xi component of velocity [m/s]
- **v** (*DataArray*) – eta component of velocity [m/s]
- **xgrid** (*xgcm.grid*) – Grid object associated with u, v
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for calculating horizontal derivatives of u and v. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for calculating horizontal derivatives of u and v. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

Returns

- *DataArray of 2D convergence of horizontal currents on rho/rho grids.*
- Output is *[T,Z,Y,X]*.

Notes

2D convergence = u_x + v_y Resource for more information: https://uw.pressbooks.pub/ocean285/chapter/the-divergence/

Examples

```
>>> ds, xgrid = xroms.roms_dataset(ds)
>>> xroms.convergence(u, v, xgrid)
```

xroms.derived.**dudz**(*u*, *xgrid*, *sboundary='extend'*, *sfill_value=None*)

Calculate the xi component of vertical shear [1/s]

Parameters

- **u** (*DataArray*) – xi component of velocity [m/s]
- **xgrid** (*xgcm.grid*) – Grid object associated with u
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for calculating z derivative. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary fill value associated with sboundary input. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

Returns

- *DataArray of xi component of vertical shear on u/w grids.*
- Output is *[T,Z,Y,X]*.

Notes

u_z = ddz(u) Wrapper of *ddz*

Examples

```
>>> xroms.dudz(u, xgrid)
```

xroms.derived.**dvdz**(*v*, *xgrid*, *sboundary='extend'*, *sfill_value=None*)

Calculate the eta component of vertical shear [1/s]

Parameters

- **v** (*DataArray*) – eta component of velocity [m/s]
- **xgrid** (*xgcm.grid*) – Grid object associated with v

- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for calculating z derivative. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary fill value associated with sboundary input. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

Returns

- *DataArray of eta component of vertical shear on v/w grids.*
- Output is *[T,Z,Y,X]*.

Notes

v_z = ddz(v) Wrapper of *ddz*

Examples

```
>>> xroms.dvdz(v, xgrid)
```

xroms.derived.**ertel**(*phi*, *u*, *v*, *f*, *xgrid*, *hcoord='rho'*, *scoord='s_rho'*, *hboundary='extend'*, *hfill_value=None*, *sboundary='extend'*, *sfill_value=None*)

Calculate Ertel potential vorticity of phi.

Parameters

- **phi** (*DataArray*) – Conservative tracer. Usually this would be the buoyancy but could be another approximately conservative tracer. The buoyancy can be calculated as: >>> xroms.buoyancy(temp, salt, 0) and then input as *phi*.
- **u** (*DataArray*) – xi component of velocity [m/s]
- **v** (*DataArray*) – eta component of velocity [m/s]
- **f** (*DataArray*) – Coriolis parameter [1/s]
- **xgrid** (*xgcm.grid*) – Grid object associated with u, v
- **hcoord** (*string, optional.*) – Name of horizontal grid to interpolate output to. Options are 'rho', 'psi', 'u', 'v'.
- **scoord** (*string, optional.*) – Name of vertical grid to interpolate output to. Options are 's_rho', 's_w', 'rho', 'w'.
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for calculating horizontal derivatives of phi and for calculating relative vorticity. This same value will be used for all horizontal grid changes too. From xgcm documentation: A flag indicating how to handle boundaries:

  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for calculating horizontal and vertical derivatives of phi, and for calculating relative vorticity. This same value will be used for all vertical grid changes too. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

Returns

- *DataArray of the Ertel potential vorticity for the input tracer.*
- Output is *[T,Z,Y,X]*.

Notes

epv = -v_z * phi_x + u_z * phi_y + (f + v_x - u_y) * phi_z

This is not set up to accept different boundary choices for different variables.

Example usage: >>> xroms.ertel(ds.dye_01, ds.u, ds.v, ds.f, xgrid, scoord='s_w');

`xroms.derived.omega(u, v)`

Calculate s-grid vertical velocity from u and v [m/s]

TO BE INPUT BY VRX.

Parameters

- **u** (*DataArray*) – xi component of velocity [m/s]
- **v** (*DataArray*) – eta component of velocity [m/s]

Returns

- *DataArray of vertical component of velocity with respect to the s grid*
- *on [horizontal]/[vertical] grids.*
- Output is *[T,Z,Y,X]*.

Notes

[Give calculation]

Examples

```
>>> xroms.omega(u, v)
```

xroms.derived.**relative_vorticity**(*u*, *v*, *xgrid*, *hboundary='extend'*, *hfill_value=None*, *sboundary='extend'*, *sfill_value=None*)

Calculate the vertical component of the relative vorticity [1/s]

Parameters

- **u** (*DataArray*) – xi component of velocity [m/s]
- **v** (*DataArray*) – eta component of velocity [m/s]
- **xgrid** (*xgcm.grid*) – Grid object associated with u, v
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for calculating horizontal derivatives of u and v. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for calculating horizontal derivatives of u and v. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

Returns

- *DataArray of vertical component of relative vorticity psi/w grids.*
- Output is *[T,Z,Y,X]*.

Notes

relative_vorticity = v_x - u_y

Examples

```
>>> xroms.relative_vorticity(u, v, xgrid)
```

xroms.derived.**speed**(*u*, *v*, *xgrid*, *hboundary='extend'*, *hfill_value=None*)

Calculate horizontal speed [m/s] from u and v components

Parameters

- **u** (*DataArray*) – xi component of velocity [m/s]
- **v** (*DataArray*) – eta component of velocity [m/s]
- **xgrid** (*xgcm.grid*) – Grid object associated with u, v
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for moving to rho grid. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary fill value selection for moving to rho grid. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

Returns

- *DataArray of speed calculated on rho/rho grids.*
- Output is *[T,Z,Y,X]*.

Notes

speed = np.sqrt(u^2 + v^2)

xroms.derived.**uv_geostrophic**(*zeta*, *f*, *xgrid*, *hboundary='extend'*, *hfill_value=None*, *which='both'*)

Calculate geostrophic velocities from zeta [m/s]

Parameters

- **zeta** (*DataArray*) – sea surface height [m]
- **f** (*DataArray or ndarray*) – Coriolis parameter [1/s]
- **xgrid** (*xgcm.grid*) – Grid object associated with zeta
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for moving f to rho grid. From xgcm documentation: A flag indicating how to handle boundaries:

  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection for moving f to rho grid. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **which** (*string, optional*) – Which components of geostrophic velocity to return.
  - 'both': return both components of hgrad
  - 'xi': return only xi-direction.
  - 'eta': return only eta-direction.

Returns

- *DataArrays of components of geostrophic velocity*
- *calculated on their respective grids.*
- Output is *[T,Y,X]*.

Notes

ug = -g * zeta_eta / (d eta * f) # on u grid

vg = g * zeta_xi / (d xi * f) # on v grid

Translation to Python of Matlab copy of surf_geostr_vel of IRD Roms_Tools.

Good resourcefor more information: https://uw.pressbooks.pub/ocean285/chapter/geostrophic-balance/

Examples

```
>>> xroms.uv_geostrophic(ds.zeta, ds.f, xgrid)
```

xroms.derived.**vertical_shear**(*dudz*, *dvdz*, *xgrid*, *hboundary='extend'*, *hfill_value=None*)

Calculate the vertical shear [1/s]

Parameters

- **dudz** (*DataArray*) – xi component of vertical shear [1/s]
- **dvdz** (*DataArray*) – eta compoenent of vertical shear [1/s]
- **xgrid** (*xgcm.grid*) – Grid object associated with dudz, dvdz
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for moving dudz and dvdz to rho grid. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)

- 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection for moving to rho grid. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

Returns

- *DataArray of vertical shear on rho/w grids.*
- Output is *[T,Z,Y,X]*.

Notes

vertical_shear = np.sqrt(u_z^2 + v_z^2)

Examples

```
>>> xroms.vertical_shear(dudz, dvdz, xgrid)
```

xroms.derived.**w**(*u*, *v*)

Calculate vertical velocity from u and v [m/s]

TO BE INPUT BY VRX.

Parameters

- **u** (*DataArray*) – xi component of velocity [m/s]
- **v** (*DataArray*) – eta component of velocity [m/s]

Returns

- *DataArray of vertical component of velocity on [horizontal]/[vertical] grids.*
- Output is *[T,Z,Y,X]*.

Notes

[Give calculation]

Examples

```
>>> xroms.w(u, v)
```

1.6.3 xroms.interp

Interpolation functions.

Functions

interpll(var, lons, lats[, which])	Interpolate var to lons/lats positions.
isoslice(var, iso_values, xgrid[, ...])	Interpolate var to iso_values.

xroms.interp.**interpll**(*var*, *lons*, *lats*, *which='pairs'*, ***kwargs*)

Interpolate var to lons/lats positions.

Wraps xESMF to perform proper horizontal interpolation on non-flat Earth.

Parameters

- **var** (*DataArray*) – Variable to operate on.
- **lons** (*list, ndarray*) – Longitudes to interpolate to. Will be flattened upon input.
- **lats** (*list, ndarray*) – Latitudes to interpolate to. Will be flattened upon input.
- **which** (*str, optional*) – Which type of interpolation to do:
  - "pairs": lons/lats as unstructured coordinate pairs (in xESMF language, LocStream).
  - "grid": 2D array of points with 1 dimension the lons and the other dimension the lats.
- ****kwargs** – passed on to xESMF Regridder class

Returns

- *DataArray of var interpolated to lons/lats. Dimensionality will be the*
- *same as var except the Y and X dimensions will be 1 dimension called*
- *"locations" that lons.size if which=='pairs', or 2 dimensions called*
- *"lat" and "lon" if which=='grid' that are of lats.size and lons.size,*
- *respectively.*

Notes

var cannot have chunks in the Y or X dimensions.

cf-xarray should still be usable after calling this function.

Examples

To return 1D pairs of points, in this case 3 points:

```
>>> xroms.interpll(var, [-96, -97, -96.5], [26.5, 27, 26.5], which='pairs')
```

To return 2D pairs of points, in this case a 3x3 array of points:

```
>>> xroms.interpll(var, [-96, -97, -96.5], [26.5, 27, 26.5], which='grid')
```

xroms.interp.**isoslice**(*var*, *iso_values*, *xgrid*, *iso_array=None*, *axis='Z'*)

Interpolate var to iso_values.

This wraps *xgcm transform* function for slice interpolation, though *transform* has additional functionality.

Parameters

- **var** (*DataArray*) – Variable to operate on.
- **iso_values** (*list, ndarray*) – Values to interpolate to. If calculating var at fixed depths, iso_values are the fixed depths, which should be negative if below mean sea level. If input as array, should be 1D.
- **xgrid** (*xgcm.grid, optional*) – Grid object associated with var.
- **iso_array** (*DataArray, optional*) – Array that var is interpolated onto (e.g., z coordinates or density). If calculating var on fixed depth slices, iso_array contains the depths [m] associated with var. In that case and if None, will use z coordinate attached to var. Also use this option if you want to interpolate with z depths constant in time and input the appropriate z coordinate.
- **dim** (*str, optional*) – Dimension over which to calculate isoslice. If calculating var onto fixed depths, *dim='Z'*. Options are 'Z', 'Y', and 'X'.

Returns

- *DataArray of var interpolated to iso_values. Dimensionality will be the*
- *same as var except with dim dimension of size of iso_values.*

Notes

var cannot have chunks in the dimension dim.

cf-xarray should still be usable after calling this function.

Examples

To calculate temperature onto fixed depths:

```
>>> xroms.isoslice(ds.temp, np.linspace(0, -30, 50))
```

To calculate temperature onto salinity:

```
>>> xroms.isoslice(ds.temp, np.arange(0, 36), iso_array=ds.salt, axis='Z')
```

Calculate lat-z slice of salinity along a constant longitude value (-91.5):

```
>>> xroms.isoslice(ds.salt, -91.5, iso_array=ds.lon_rho, axis='X')
```

Calculate slice of salt at 28 deg latitude

```
>>> xroms.isoslice(ds.salt, 28, iso_array=ds.lat_rho, axis='Y')
```

Interpolate temp to salinity values between 0 and 36 in the X direction

```
>>> xroms.isoslice(ds.temp, np.linspace(0, 36, 50), iso_array=ds.salt, axis='X')
```

Interpolate temp to salinity values between 0 and 36 in the Z direction

```
>>> xroms.isoslice(ds.temp, np.linspace(0, 36, 50), iso_array=ds.salt, axis='Z')
```

Calculate the depth of a specific isohaline (33):

```
>>> xroms.isoslice(ds.salt, 33, iso_array=ds.z_rho, axis='Z')
```

Calculate dye 10 meters above seabed. Either do this on the vertical rho grid, or first change to the w grid and then use *isoslice*. You may prefer to do the latter if there is a possibility that the distance above the seabed you are interpolating to (10 m) could be below the deepest rho grid depth.

- on rho grid directly:

```
>>> height_from_seabed = ds.z_rho + ds.h
>>> height_from_seabed.name = 'z_rho'
>>> xroms.isoslice(ds.dye_01, 10, iso_array=height_from_seabed, axis='Z')
```

- on w grid:

```
>>> var_w = ds.dye_01.xroms.to_grid(scoord='w').chunk({'s_w': -1})
>>> ds['dye_01_w'] = var_w  # currently this is the easiest way to reattached␣
↪coords xgcm variables
>>> height_from_seabed = ds.z_w + ds.h
>>> height_from_seabed.name = 'z_w'
>>> xroms.isoslice(ds['dye_01_w'], 10, iso_array=height_from_seabed, axis='Z')
```

1.6.4 xroms.roms_seawater

Functions related to density of seawater.

Functions

M2(rho, xgrid[, rho0, hboundary, ...])	Calculate the horizontal buoyancy gradient.
N2(rho, xgrid[, rho0, sboundary, sfill_value])	Calculate buoyancy frequency squared (vertical buoyancy gradient).
buoyancy(sig0[, rho0])	Calculate buoyancy [m/s^2] based on potential density.
density(temp, salt[, z])	Calculate the density [kg/m^3] as calculated in ROMS.
mld(sig0, xgrid, h, mask[, z, thresh])	Calculate the mixed layer depth [m], return positive and as depth if no value calculated.
potential_density(temp, salt[, z])	Calculate potential density [kg/m^3] with constant depth reference.

xroms.roms_seawater.**M2**(*rho*, *xgrid*, *rho0=1025.0*, *hboundary='extend'*, *hfill_value=None*, *sboundary='fill'*, *sfill_value=nan*, *z=None*)

Calculate the horizontal buoyancy gradient.

Parameters

- **rho** (*DataArray*) – Density [kg/m^3]
- **xgrid** (*xgcm.grid*) – Grid object associated with rho
- **rho0** (*int, float, optional*) – Reference density [kg/m^3].

- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for calculating horizontal derivatives of rho. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for calculating horizontal derivatives of rho. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary fill value associated with sboundary input. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **z** (*DataArray, optional*) – Depths [m] associated with rho. If None, use z coordinate attached to temperature.

Returns

- *DataArray of the horizontal buoyancy gradient on rho/w grids.*
- Output is *[T,Z,Y,X]*.

Notes

M2 = g/rho0 * sqrt(d(rho)/dxi^2 + d(rho)deta^2)

g=9.81 [m/s^2]

Examples

```
>>> xroms.M2(rho, xgrid)
```

xroms.roms_seawater.**N2**(*rho*, *xgrid*, *rho0=1025.0*, *sboundary='fill'*, *sfill_value=nan*)

Calculate buoyancy frequency squared (vertical buoyancy gradient).

Parameters

- **rho** (*DataArray*) – Density [kg/m^3]

- **xgrid** (*xgcm.grid*) – Grid object associated with rho
- **rho0** (*int, float*) – Reference density [kg/m^3].
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for calculating z derivative. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary fill value associated with sboundary input. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

Returns

- *DataArray of buoyancy frequency squared on rho/w grids.*
- Output is *[T,Z,Y,X]*.

Notes

N2 = -g d(rho)/dz / rho0

Examples

```
>>> xroms.N2(rho, xgrid)
```

xroms.roms_seawater.**buoyancy**(*sig0*, *rho0=1025.0*)

Calculate buoyancy [m/s^2] based on potential density.

Parameters

- **sig0** (*DataArray, ndarray*) – Potential density [kg/m^3]
- **rho0** (*int, float, optional*) – Reference density [kg/m^3].

Returns

- *DataArray or ndarray of calculated buoyancy on rho/rho grids.*
- Output is *[T,Z,Y,X]*.

Notes

buoyancy = -g * rho / rho0

Uses equation of state based on ROMS Nonlinear/rho_eos.F

g=9.81 [m/s^2]

Examples

```
>>> xroms.potential_density(ds.temp, ds.salt)
```

xroms.roms_seawater.**density**(*temp*, *salt*, *z=None*)

Calculate the density [kg/m^3] as calculated in ROMS.

Parameters

- **temp** (*DataArray, ndarray*) – Temperature [Celsius]
- **salt** (*DataArray, ndarray*) – Salinity
- **z** (*DataArray, ndarray, int, float, optional*) – Depth [m]. To specify a reference depth, use a constant. If None, use z coordinate attached to temperature.

Returns

- *DataArray or ndarray of calculated density on rho/rho grids.*
- Output is *[T,Z,Y,X]*.

Notes

Equation of state based on ROMS Nonlinear/rho_eos.F

Examples

```
>>> xroms.density(ds.temp, ds.salt)
```

xroms.roms_seawater.**mld**(*sig0*, *xgrid*, *h*, *mask*, *z=None*, *thresh=0.03*)

Calculate the mixed layer depth [m], return positive and as depth if no value calculated.

Parameters

- **sig0** (*DataArray*) – Potential density [kg/m^3]
- **xgrid** – xgcm grid
- **h** (*DataArray, ndarray*) – Depths [m].
- **mask** (*DataArray, ndarray*) – mask to match sig0
- **z** (*DataArray, ndarray, optional*) – The vertical depths associated with sig0. Should be on 'rho' grid horizontally and vertically. Use z coords associated with DataArray sig0 if not input.
- **thresh** (*float, optional*) – For detection of mixed layer [kg/m^3]

Returns

- *DataArray of mixed layer depth on rho horizontal grid.*

- Output is *[T,Y,X]*.

Notes

Mixed layer depth is based on the fixed Potential Density (PD) threshold.

Converted to xroms by K. Thyng Aug 2020 from:

Update history: v1.0 DL 2020Jun07

References: ncl mixed_layer_depth function at https://github.com/NCAR/ncl/blob/ed6016bf579f8c8e8f77341503daef3c532f1069/ni/src/lib/nfpfort/ocean.f de Boyer Montégut, C., Madec, G., Fischer, A. S., Lazar, A., & Iudicone, D. (2004). Mixed layer depth over the global ocean: An examination of profile data and a profile-based climatology. Journal of Geophysical Research: Oceans, 109(C12).

Useful resources:

- Climate Data Toolbox documentation: https://www.chadagreene.com/CDT/mld_documentation.html
- MLD calculation from MDTF: https://github.com/NOAA-GFDL/MDTF-diagnostics/blob/437d30590c45e8b7dd0cd01a3dc67066a2137115/diagnostics/mixed_layer_depth/mixed_layer_depth.py#L147

Examples

```
>>> xroms.mld(sig0, h, mask)
```

xroms.roms_seawater.**potential_density**(*temp*, *salt*, *z=0*)

Calculate potential density [kg/m^3] with constant depth reference.

Parameters

- **temp** (*DataArray, ndarray*) – Temperature [Celsius]
- **salt** (*DataArray, ndarray*) – Salinity
- **z** (*int, float, optional*) – Reference depth [m].

Returns

- *DataArray or ndarray of calculated potential density on rho/rho grids.*
- Output is *[T,Z,Y,X]*.

Notes

Uses equation of state based on ROMS Nonlinear/rho_eos.F

Examples

```
>>> xroms.potential_density(ds.temp, ds.salt)
```

1.6.5 xroms.utilities

Functions that act on DataArrays or Datasets.

Functions

argsel2d(lons, lats, lon0, lat0)	Find the indices of coordinate pair closest to another point.
ddeta(var, xgrid[, z, hcoord, scoord, ...])	Calculate d/deta for a variable.
ddxi(var, xgrid[, z, hcoord, scoord, ...])	Calculate d/dxi for a variable.
ddz(var, xgrid[, hcoord, scoord, hboundary, ...])	Calculate d/dz for a variable.
grid_interp(xgrid, da, dim[, ...])	Interpolate da in dim
gridmean(var, xgrid, dim)	Calculate mean accounting for variable spatial grid.
gridsum(var, xgrid, dim)	Calculate sum accounting for variable spatial grid.
hgrad(q, xgrid[, which, z, hcoord, scoord, ...])	Return gradients of property q accounting for s coordinates.
order(var)	Reorder var to typical dimensional ordering.
sel2d(var, lons, lats, lon0, lat0)	Find the value of the var at closest location to lon0,lat0.
subset(ds[, X, Y])	Subset model output horizontally using isel, properly accounting for horizontal grids.
to_grid(var, xgrid[, hcoord, scoord, ...])	Implement grid changes.
to_psi(var, xgrid[, hboundary, hfill_value])	Change var to psi horizontal grid.
to_rho(var, xgrid[, hboundary, hfill_value])	Change var to rho horizontal grid.
to_s_rho(var, xgrid[, sboundary, sfill_value])	Change var to rho vertical grid.
to_s_w(var, xgrid[, sboundary, sfill_value])	Change var to w vertical grid.
to_u(var, xgrid[, hboundary, hfill_value])	Change var to u horizontal grid.
to_v(var, xgrid[, hboundary, hfill_value])	Change var to v horizontal grid.
xisoslice(iso_array, iso_value, ...)	Calculate an isosurface.

xroms.utilities.**argsel2d**(*lons*, *lats*, *lon0*, *lat0*)

Find the indices of coordinate pair closest to another point.

Parameters

- **lons** (*DataArray, ndarray, list*) – Longitudes of points to search through for closest point.
- **lats** (*DataArray, ndarray, list*) – Latitudes of points to search through for closest point.
- **lon0** (*float, int*) – Longitude of comparison point.
- **lat0** (*float, int*) – Latitude of comparison point.

Returns

- *Index or indices of location in coordinate pairs made up of lons, lats*
- *that is closest to location lon0, lat0. Number of dimensions of*

- *returned indices will correspond to the shape of input lons.*

Notes

This function uses Great Circle distance to calculate distances assuming longitudes and latitudes as point coordinates. Uses cartopy function *Geodesic*: https://scitools.org.uk/cartopy/docs/latest/cartopy/geodesic.html

If searching for the closest grid node to a lon/lat location, be sure to use the correct horizontal grid (rho, u, v, or psi). This is accounted for if this function is used through the accessor.

Examples

```
>>> xroms.argsel2d(ds.lon_rho, ds.lat_rho, -96, 27)
```

xroms.utilities.**ddeta**(*var*, *xgrid*, *z=None*, *hcoord=None*, *scoord=None*, *hboundary='extend'*, *hfill_value=nan*, *sboundary='extend'*, *sfill_value=nan*, *attrs=None*)

Calculate d/deta for a variable.

Note that you need the 3D metrics for horizontal derivatives for ROMS, so `include_3D_metrics=True` in `xroms.roms_dataset()`.

Parameters

- **var** (*DataArray, ndarray*) – Variable to operate on.
- **xgrid** (*xgcm.grid*) – Grid object associated with var.
- **z** (*DataArray, ndarray, optional*) – Depth [m]. If None, use z coordinate attached to var.
- **hcoord** (*string, optional*) – Name of horizontal grid to interpolate output to. Options are 'rho', 'psi', 'u', 'v'.
- **scoord** (*string, optional*) – Name of vertical grid to interpolate output to. Options are 's_rho', 's_w', 'rho', 'w'.
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for calculating horizontal derivative of var. This same value will be used for grid changes too. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for calculating horizontal derivative of var. This same value will be used for vertical grid changes too. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.

- 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
- 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **attrs** (*dict, optional*) – Dictionary of attributes to add to resultant arrays. Requires that q is DataArray. For example: *attrs={'name': 'varname', 'long_name': 'longvarname', 'units': 'units'}*

Returns

- *DataArray or ndarray of dqdeta, the gradient of q in the eta-direction with*
- *attributes altered to reflect calculation.*

Notes

dqdeta = dqdy*dzdz - dqdz*dzdy

Derivatives are taken in the ROMS curvilinear grid native eta-direction.

These derivatives properly account for the fact that ROMS vertical coordinates are s coordinates and therefore can vary in time and space.

This will alter the number of points in the eta and s dimensions.

Examples

```
>>> xroms.ddeta(ds.salt, xgrid)
```

xroms.utilities.**ddxi**(*var*, *xgrid*, *z=None*, *hcoord=None*, *scoord=None*, *hboundary='extend'*, *hfill_value=nan*, *sboundary='extend'*, *sfill_value=nan*, *attrs=None*)

Calculate d/dxi for a variable.

Note that you need the 3D metrics for horizontal derivatives for ROMS, so `include_3D_metrics=True` in `xroms.roms_dataset()`.

Parameters

- **var** (*DataArray*) – Variable to operate on.
- **xgrid** (*xgcm.grid*) – Grid object associated with var.
- **z** (*DataArray, ndarray, optional*) – Depth [m]. If None, use z coordinate attached to var.
- **hcoord** (*string, optional*) – Name of horizontal grid to interpolate output to. Options are 'rho', 'psi', 'u', 'v'.
- **scoord** (*string, optional*) – Name of vertical grid to interpolate output to. Options are 's_rho', 's_w', 'rho', 'w'.
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for calculating horizontal derivative of var. This same value will be used for all horizontal grid changes too. From xgcm documentation: A flag indicating how to handle boundaries:

  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for calculating horizontal derivative of var. This same value will be used for all vertical grid changes too. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **attrs** (*dict, optional*) – Dictionary of attributes to add to resultant arrays. Requires that q is DataArray. For example: *attrs={'name': 'varname', 'long_name': 'longvarname', 'units': 'units'}*

Returns

- *DataArray of dqdxi, the gradient of q in the xi-direction with*
- *attributes altered to reflect calculation.*

Notes

dqdxi = dqdx*dzdz - dqdz*dzdx

Derivatives are taken in the ROMS curvilinear grid native xi-direction.

These derivatives properly account for the fact that ROMS vertical coordinates are s coordinates and therefore can vary in time and space.

This will alter the number of points in the xi and s dimensions.

Examples

```
>>> xroms.ddxi(ds.salt, xgrid)
```

xroms.utilities.**ddz**(*var*, *xgrid*, *hcoord=None*, *scoord=None*, *hboundary='extend'*, *hfill_value=None*, *sboundary='extend'*, *sfill_value=None*, *attrs=None*)

Calculate d/dz for a variable.

Parameters

- **var** (*DataArray*) – Variable to operate on.
- **xgrid** (*xgcm.grid*) – Grid object associated with var
- **hcoord** (*string, optional.*) – Name of horizontal grid to interpolate output to. Options are 'rho', 'psi', 'u', 'v'.
- **scoord** (*string, optional.*) – Name of vertical grid to interpolate output to. Options are 's_rho', 's_w', 'rho', 'w'.
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for calculating horizontal derivative of var. This same value will be used for grid changes too. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for calculating z derivative. This same value will be used for grid changes too. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary fill value associated with sboundary input. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **attrs** (*dict, optional*) – Dictionary of attributes to add to resultant arrays. Requires that q is DataArray. For example: *attrs={'name': 'varname', 'long_name': 'longvarname', 'units': 'units'}*

Returns

- *DataArray of vertical derivative of variable with*
- *attributes altered to reflect calculation.*

Notes

This will alter the number of points in the s dimension.

Examples

```
>>> xroms.ddz(ds.salt, xgrid)
```

xroms.utilities.**grid_interp**(*xgrid*, *da*, *dim*, *which_xgcm_function='interp'*, ***kwargs*)

Interpolate da in dim

This function is necessary because of weirdness with chunking with xgcm. More info: https://github.com/xgcm/xgcm/issues/522

Parameters

- **xgrid** (*xgcm grid object*) – _description_
- **da** (*DataArray*) – interpolating from this dataarray
- **dim** (*str*) – interpolating grids in this dimension
- **which_xgcm_function** (*"interp"*) – But could instead be "integrate"

Returns

interpolated down one dimension in dim

Return type

DataArray

xroms.utilities.**gridmean**(*var*, *xgrid*, *dim*)

Calculate mean accounting for variable spatial grid.

Parameters

- **var** (*DataArray or ndarray*) – Variable to operate on.
- **xgrid** (*xgcm.grid*) – Grid object associated with var
- **dim** (*str, list, tuple*) – Spatial dimension names to average over. In the *xgcm* convention, the allowable names are 'Z', 'Y', or 'X'.

Returns

- *DataArray or ndarray of average calculated over dim accounting*
- *for variable spatial grid.*

Notes

If result is DataArray, long name attribute is modified to describe calculation.

Examples

Note that the following two approaches are equivalent:

```
>>> app1 = xroms.gridmean(ds.u, xgrid, ('Y','X'))
>>> app2 = (ds.u*ds.dy_u*ds.dx_u).sum(('eta_rho','xi_u'))/(ds.dy_u*ds.dx_u).sum((
↪'eta_rho','xi_u'))
>>> np.allclose(app1, app2)
```

xroms.utilities.**gridsum**(*var*, *xgrid*, *dim*)

Calculate sum accounting for variable spatial grid.

Parameters

- **var** (*DataArray or ndarray*) – Variable to operate on.
- **xgrid** (*xgcm.grid*) – Grid object associated with var
- **dim** (*str, list, tuple*) – Spatial dimension names to sum over. In the *xgcm* convention, the allowable names are 'Z', 'Y', or 'X'.

Returns

- *DataArray or ndarray of sum calculated over dim accounting*
- *for variable spatial grid.*

Notes

If result is DataArray, long name attribute is modified to describe calculation.

Examples

Note that the following two approaches are equivalent:

```
>>> app1 = xroms.gridsum(ds.u, xgrid, ('Z','X'))
>>> app2 = (ds.u*ds.dz_u * ds.dx_u).sum(('s_rho','xi_u'))
>>> np.allclose(app1, app2)
```

xroms.utilities.**hgrad**(*q*, *xgrid*, *which='both'*, *z=None*, *hcoord=None*, *scoord=None*, *hboundary='extend'*, *hfill_value=None*, *sboundary='extend'*, *sfill_value=None*, *attrs=None*)

Return gradients of property q accounting for s coordinates.

Note that you need the 3D metrics for horizontal derivatives for ROMS, so `include_3D_metrics=True` in `xroms.roms_dataset()`.

Parameters

- **q** (*DataArray*) – Property to take gradients of.
- **xgrid** (*xgcm.grid*) – Grid object associated with q.
- **which** (*string, optional*) – Which components of gradient to return.
  - 'both': return both components of hgrad.
  - 'xi': return only xi-direction.
  - 'eta': return only eta-direction.

- **z** (*DataArray, ndarray, optional*) – Depth [m]. If None, use z coordinate attached to q.
- **hcoord** (*string, optional*) – Name of horizontal grid to interpolate output to. Options are 'rho', 'psi', 'u', 'v'.
- **scoord** (*string, optional*) – Name of vertical grid to interpolate output to. Options are 's_rho', 's_w', 'rho', 'w'.
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for calculating horizontal derivatives of q. This same value will be used for all horizontal grid changes too. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for calculating horizontal derivatives of q. This same value will be used for all vertical grid changes too. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **attrs** (*dict, optional*) – Dictionary of attributes to add to resultant arrays. Requires that q is DataArray.

Returns

- *DataArray(s) of dqdxi and/or dqdeta, the gradients of q*
- *in the xi- and eta-directions with attributes altered to reflect calculation.*

Notes

dqdxi = dqdx*dzdz - dqdz*dzdx

dqdeta = dqdy*dzdz - dqdz*dzdy

Derivatives are taken in the ROMS curvilinear grid native xi- and eta- directions.

These derivatives properly account for the fact that ROMS vertical coordinates are s coordinates and therefore can vary in time and space.

The xi derivative will alter the number of points in the xi and s dimensions. The eta derivative will alter the number of points in the eta and s dimensions.

Examples

```
>>> dtempdxi, dtempdeta = xroms.hgrad(ds.temp, xgrid)
```

xroms.utilities.**order**(*var*)

Reorder var to typical dimensional ordering.

Parameters
var (*DataArray*) – Variable to operate on.

Returns

- *DataArray with dimensional order ['T', 'Z', 'Y', 'X'], or whatever subset of*
- *dimensions are present in var.*

Notes

Do not consider previously-selected dimensions that are kept on as coordinates but cannot be transposed anymore. This is accomplished with *.reset_coords(drop=True)*.

Examples

```
>>> xroms.order(var)
```

xroms.utilities.**sel2d**(*var*, *lons*, *lats*, *lon0*, *lat0*)

Find the value of the var at closest location to lon0,lat0.

Parameters

- **var** (*DataArray, ndarray*) – Variable to operate on.
- **lons** (*DataArray, ndarray, list*) – Longitudes of points to search through for closest point.
- **lats** (*DataArray, ndarray, list*) – Latitudes of points to search through for closest point.
- **lon0** (*float, int*) – Longitude of comparison point.
- **lat0** (*float, int*) – Latitude of comparison point.

Returns

- *Value in var of location in coordinate pairs made up of lons, lats*

- *that is closest to location lon0, lat0. If var has other*
- *dimensions, they are brought along.*

Notes

This function uses Great Circle distance to calculate distances assuming longitudes and latitudes as point coordinates. Uses cartopy function *Geodesic*: https://scitools.org.uk/cartopy/docs/latest/cartopy/geodesic.html

If searching for the closest grid node to a lon/lat location, be sure to use the correct horizontal grid (rho, u, v, or psi). This is accounted for if this function is used through the accessor.

This is meant to be used by the accessor to conveniently wrap *argsel2d*.

Examples

```
>>> xroms.sel2d(ds.temp, ds.lon_rho, ds.lat_rho, -96, 27)
```

xroms.utilities.**subset**(*ds*, *X=None*, *Y=None*)

Subset model output horizontally using isel, properly accounting for horizontal grids.

Parameters

- **ds** (*xarray Dataset*) – Dataset of ROMS model output. Assumes that full regular grid setup is available and has been read in using xroms so that dimension names have been updated.
- **X** (*slice, optional*) – Slice in X dimension using form *X=slice(start, stop, step)*. For example,

  ```
  >>> X=slice(20,40,2)
  ```

  Indices are used for rho grid, and psi grid is reduced accordingly.
- **Y** (*slice, optional*) – Slice in Y dimension using form *Y=slice(start, stop, step)*. For example,

  ```
  >>> Y=slice(20,40,2)
  ```

  Indices are used for rho grid, and psi grid is reduced accordingly.

Returns

- *Dataset with form as if model had been run at the subsetted size. That is, the outermost*
- *cells of the rho grid are like ghost cells and the psi grid is one inward from this size*
- *in each direction.*

Notes

X and Y must be slices, not single numbers.

Examples

Subset only in Y direction:

```
>>> xroms.subset(ds, Y=slice(50,100))
```

Subset in X and Y:

```
>>> xroms.subset(ds, X=slice(20,40), Y=slice(50,100))
```

xroms.utilities.**to_grid**(*var*, *xgrid*, *hcoord=None*, *scoord=None*, *hboundary='extend'*, *hfill_value=None*, *sboundary='extend'*, *sfill_value=None*, *attrs=None*)

Implement grid changes.

Parameters

- **var** (*DataArray or ndarray*) – Variable to operate on.
- **xgrid** (*xgcm.grid*) – Grid object associated with var
- **hcoord** (*string, optional.*) – Name of horizontal grid to interpolate output to. Options are 'rho', 'psi', 'u', 'v'.
- **scoord** (*string, optional.*) – Name of vertical grid to interpolate output to. Options are 's_rho', 's_w', 'rho', 'w'.
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for grid changes. From xgcm documentation:

  A flag indicating how to handle boundaries: * None: Do not apply any boundary conditions. Raise an error if

  boundary conditions are required for the operation.

  – 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)

  – 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for grid changes. From xgcm documentation: A flag indicating how to handle boundaries:

  – None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.

  – 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)

  – 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

Returns

- *DataArray or ndarray interpolated onto hcoord horizontal and scoord*
- *vertical grids.*

Notes

If var is already on selected grid, nothing happens.

Examples

```
>>> xroms.to_grid(ds.salt, xgrid, hcoord='rho', scoord='w')
```

xroms.utilities.**to_psi**(*var*, *xgrid*, *hboundary='extend'*, *hfill_value=None*)

Change var to psi horizontal grid.

Parameters

- **var** (*DataArray or ndarray*) – Variable to operate on.
- **xgrid** (*xgcm.grid*) – Grid object associated with var
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for grid changes. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

Return type

DataArray or ndarray interpolated onto psi horizontal grid.

Notes

If var is already on psi grid, nothing happens.

to_grid function wraps all of the *to_** functions so one function can be used for all grid changes.

Examples

```
>>> xroms.to_psi('salt', xgrid)
```

xroms.utilities.**to_rho**(*var*, *xgrid*, *hboundary='extend'*, *hfill_value=None*)

Change var to rho horizontal grid.

Parameters

- **var** (*DataArray or ndarray*) – Variable to operate on.
- **xgrid** (*xgcm.grid*) – Grid object associated with var
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for grid changes. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

Return type

DataArray or ndarray interpolated onto rho horizontal grid.

Notes

If var is already on rho grid, nothing happens.

to_grid function wraps all of the *to_** functions so one function can be used for all grid changes.

Examples

```
>>> xroms.to_rho('salt', xgrid)
```

xroms.utilities.**to_s_rho**(*var*, *xgrid*, *sboundary='extend'*, *sfill_value=None*)

Change var to rho vertical grid.

Parameters

- **var** (*DataArray or ndarray*) – Variable to operate on.
- **xgrid** (*xgcm.grid*) – Grid object associated with var
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for grid changes. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.

  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

Return type
DataArray or ndarray interpolated onto rho vertical grid.

Notes

If var is already on rho grid, nothing happens.

to_grid function wraps all of the *to_** functions so one function can be used for all grid changes.

Examples

```
>>> xroms.to_s_rho('salt', xgrid)
```

xroms.utilities.**to_s_w**(*var*, *xgrid*, *sboundary='extend'*, *sfill_value=None*)

Change var to w vertical grid.

Parameters

- **var** (*DataArray or ndarray*) – Variable to operate on.
- **xgrid** (*xgcm.grid*) – Grid object associated with var
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for grid changes. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

Return type
DataArray or ndarray interpolated onto rho vertical grid.

Notes

If var is already on w grid, nothing happens.

to_grid function wraps all of the *to_** functions so one function can be used for all grid changes.

Examples

```
>>> xroms.to_s_w('salt', xgrid)
```

xroms.utilities.**to_u**(*var*, *xgrid*, *hboundary='extend'*, *hfill_value=None*)

Change var to u horizontal grid.

Parameters

- **var** (*DataArray or ndarray*) – Variable to operate on.
- **xgrid** (*xgcm.grid*) – Grid object associated with var
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for grid changes. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

Return type

DataArray or ndarray interpolated onto u horizontal grid.

Notes

If var is already on u grid, nothing happens.

to_grid function wraps all of the *to_** functions so one function can be used for all grid changes.

Examples

```
>>> xroms.to_u('salt', xgrid)
```

xroms.utilities.**to_v**(*var*, *xgrid*, *hboundary='extend'*, *hfill_value=None*)

Change var to v horizontal grid.

Parameters

- **var** (*DataArray or ndarray*) – Variable to operate on.
- **xgrid** (*xgcm.grid*) – Grid object associated with var

- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for grid changes. From xgcm documentation: A flag indicating how to handle boundaries:
  - None: Do not apply any boundary conditions. Raise an error if boundary conditions are required for the operation.
  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

Return type
DataArray or ndarray interpolated onto v horizontal grid.

Notes

If var is already on v grid, nothing happens.

to_grid function wraps all of the *to_** functions so one function can be used for all grid changes.

Examples

```
>>> xroms.to_v('salt', xgrid)
```

xroms.utilities.**xisoslice**(*iso_array*, *iso_value*, *projected_array*, *coord*)

Calculate an isosurface.

This function has been possibly superseded by isoslice that wraps *xgcm.grid.transform* for the following reasons, but more testing is needed:

- The implementation of *xgcm.grid.transform* is more robust than *xisoslice* which has extra code for in case iso_value is exactly in iso_array.
- For a 5-day model file, the run time for the same call for was approximately the same for xisolice and isoslice.
- isoslice might be more computationally robust for not breaking mid-way, but this is still unclear.

This function calculates the value of projected_array on an isosurface in the array iso_array defined by iso_value.

Parameters

- **iso_array** (*DataArray, ndarray*) – Array in which the isosurface is defined
- **iso_value** (*float*) – Value of the isosurface in iso_array
- **projected_array** (*DataArray, ndarray*) – Array in which to project values on the isosurface. This can have multiple time outputs. Needs to be broadcastable from iso_array?
- **coord** (*string*) – Name of coordinate associated with the dimension along which to project

Return type
DataArray or ndarray of values of projected_array on the isosurface

Notes

Performs lazy evaluation.

xisoslice requires that iso_array be monotonic. If iso_value is not monotonic it will still run but values may be incorrect where not monotonic. If iso_value is exactly in iso_array or the value is passed twice in iso_array, a message will be printed. iso_value is changed a tiny amount in this case to account for it being in iso_array exactly. The latter case is not deal with.

Examples

Calculate lat-z slice of salinity along a constant longitude value (-91.5):

```
>>> sl = xroms.utilities.xisoslice(ds.lon_rho, -91.5, ds.salt, 'xi_rho')
```

Calculate a lon-lat slice at a constant z value (-10):

```
>>> sl = xroms.utilities.xisoslice(ds.z_rho, -10, ds.temp, 's_rho')
```

Calculate a lon-lat slice at a constant z value (-10) but without zeta changing in time:

(use ds.z_rho0 which is relative to mean sea level and does not vary in time) >>> sl = xroms.utilities.xisoslice(ds.z_rho0, -10, ds.temp, 's_rho')

Calculate the depth of a specific isohaline (33):

```
>>> sl = xroms.utilities.xisoslice(ds.salt, 33, ds.z_rho, 's_rho')
```

Calculate the salt 10 meters above the seabed. Either do this on the vertical rho grid, or first change to the w grid and then use *xisoslice*. You may prefer to do the latter if there is a possibility that the distance above the seabed you are interpolating to (10 m) could be below the deepest rho grid depth.

- on rho grid directly:

  ```
  >>> sl = xroms.xisoslice(ds.z_rho + ds.h, 10., ds.salt, 's_rho')
  ```

- on w grid:

  ```
  >>> var_w = xroms.to_s_w(ds.salt, ds.xroms.xgrid)
  >>> sl = xroms.xisoslice(ds.z_w + ds.h, 10., var_w, 's_w')
  ```

In addition to calculating the slices themselves, you may need to calculate related coordinates for plotting. For example, to accompany the lat-z slice, you may want the following:

calculate z values (s_rho)

```
>>> slz = xroms.utilities.xisoslice(ds.lon_rho, -91.5, ds.z_rho, 'xi_rho')
```

calculate latitude values (eta_rho)

```
>>> sllat = xroms.utilities.xisoslice(ds.lon_rho, -91.5, ds.lat_rho, 'xi_rho')
```

assign these as coords to be used in plot

```
>>> sl = sl.assign_coords(z=slz, lat=sllat)
```

points that should be masked

```
>>> slmask = xroms.utilities.xisoslice(ds.lon_rho, -91.5, ds.mask_rho, 'xi_rho')
```

drop masked values

```
>>> sl = sl.where(slmask==1, drop=True)
```

1.6.6 xroms.vector

Functions related to vectors.

Functions

rotate_vectors(x, y, angle[, isradians, ...])	Rotate vectors according to reference.

xroms.vector.**rotate_vectors**(*x*, *y*, *angle*, *isradians=True*, *reference='xaxis'*, *xgrid=None*, *hcoord='rho'*, *attrs=None*, ***kwargs*)

Rotate vectors according to reference.

Parameters

- **x** (*Union[float, np.array, xr.DataArray]*) – x component of vector to be rotated
- **y** (*Union[float, np.array, xr.DataArray]*) – y component of vector to be rotated
- **angle** (*Union[float,np.array,xr.DataArray]*) – Angle by which to rotate x and y.
- **isradians** (*bool, optional*) – True if angle is in radians, False for degrees, by default True
- **reference** (*str, optional*) – Which reference is angle coming from? "xaxis" if angle is the angle between the x-axis and x (positive going counter clockwise, 0 at the x axis), or "compass" if angle is 0 at north on a compass and is positive going clockwise, by default "xaxis".
- **xgrid** (*Optional[xgcm.grid.Grid], optional*) – xgcm grid, by default None. If not input, any grid changing using hcoord or kwargs is ignored.
- **hcoord** (*string, optional.*) – Name of horizontal grid to interpolate output to. Options are 'rho', 'psi', 'u', 'v'. Default 'rho'.
- **attrs** (*Optional[dict], optional*) – Dict containing two keys, "x" and "y", each a dict of attributes, by default None. Attributes should include "name", "standard_name", "long_name", "units", if possible. "name" is required.
- **kwargs** – will be passed on to *xroms.to_grid()*.

Returns

x and y, rotated by angle.

Return type

Tuple[xr.DataArray]

1.6.7 xroms.accessor

This is an accessor to xarray. It is basically a convenient way to use some of the xroms functions, which has bookkeeping in the background where possible. No functions are available only here; this connects to functions in other files.

Classes

xromsDataArrayAccessor(da)	Accessor for DataArrays.
xromsDatasetAccessor(ds)	Accessor for Datasets.

class xroms.accessor.**xromsDataArrayAccessor**(*da*)

Bases: object

Accessor for DataArrays.

Methods

argsel2d(lon0, lat0)	Find the indices of coordinate pair closest to another point.
ddeta(xgrid[, hcoord, scoord, hboundary, ...])	Calculate d/deta for a variable.
ddxi(xgrid[, hcoord, scoord, hboundary, ...])	Calculate d/dxi for variable.
ddz(xgrid[, hcoord, scoord, hboundary, ...])	Calculate d/dz for a variable.
gridmean(xgrid, dim)	Calculate mean accounting for variable spatial grid.
gridsum(xgrid, dim)	Calculate sum accounting for variable spatial grid.
interpll(lons, lats[, which])	Interpolate var to lons/lats positions.
order()	Reorder self to typical dimensional ordering.
sel2d(lon0, lat0)	Find the value of the var at closest location to lon0,lat0.
to_grid(xgrid[, hcoord, scoord, hboundary, ...])	Implement grid changes.
zslice(xgrid, depths[, z])	Interpolate var to depths.

argsel2d(*lon0*, *lat0*)

Find the indices of coordinate pair closest to another point.

Parameters

- **lon0** (*float, int*) – Longitude of comparison point.
- **lat0** (*float, int*) – Latitude of comparison point.

Return type

Indices in eta, xi of closest location to lon0, lat0.

Notes

This function uses Great Circle distance to calculate distances assuming longitudes and latitudes as point coordinates. Uses cartopy function *Geodesic*: https://scitools.org.uk/cartopy/docs/latest/cartopy/geodesic.html

Examples

```
>>> ds.temp.xroms.argsel2d(-96, 27)
```

ddeta(*xgrid*, *hcoord=None*, *scoord=None*, *hboundary='extend'*, *hfill_value=None*, *sboundary='extend'*, *sfill_value=None*, *attrs=None*)

Calculate d/deta for a variable.

Parameters

- **xgrid** – xgcm grid
- **hcoord** (*string, optional.*) – Name of horizontal grid to interpolate output to. Options are 'rho', 'psi', 'u', 'v'.
- **scoord** (*string, optional.*) – Name of vertical grid to interpolate output to. Options are 's_rho', 's_w', 'rho', 'w'.
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for calculating horizontal derivative of var. This same value will be used for grid changes too. From xgcm documentation: A flag indicating how to handle boundaries: * None: Do not apply any boundary conditions. Raise an error if

  boundary conditions are required for the operation.

  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for calculating horizontal derivative of var. This same value will be used for grid changes too. From xgcm documentation: A flag indicating how to handle boundaries: * None: Do not apply any boundary conditions. Raise an error if

  boundary conditions are required for the operation.

  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

- **attrs** (*dict, optional*) – Dictionary of attributes to add to resultant arrays. Requires that q is DataArray. For example: *attrs={'name': 'varname', 'long_name': 'longvarname', 'units': 'units'}*

Returns

- *DataArray of dqdeta, the gradient of q in the eta-direction with*
- *attributes altered to reflect calculation.*

Notes

dqdeta = dqdy*dzdz - dqdz*dzdy

Derivatives are taken in the ROMS curvilinear grid native eta-direction.

These derivatives properly account for the fact that ROMS vertical coordinates are s coordinates and therefore can vary in time and space.

This will alter the number of points in the eta and s dimensions.

Examples

```
>>> ds.salt.xroms.ddeta(xgrid)
```

ddxi(*xgrid*, *hcoord=None*, *scoord=None*, *hboundary='extend'*, *hfill_value=None*, *sboundary='extend'*, *sfill_value=None*, *attrs=None*)

Calculate d/dxi for variable.

Parameters

- **xgrid** – xgcm grid
- **hcoord** (*string, optional.*) – Name of horizontal grid to interpolate output to. Options are 'rho', 'psi', 'u', 'v'.
- **scoord** (*string, optional.*) – Name of vertical grid to interpolate output to. Options are 's_rho', 's_w', 'rho', 'w'.
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for calculating horizontal derivative of var. This same value will be used for all horizontal grid changes too. From xgcm documentation: A flag indicating how to handle boundaries: * None: Do not apply any boundary conditions. Raise an error if

  boundary conditions are required for the operation.

  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for calculating horizontal derivative of var. This same value will be used for all vertical

grid changes too. From xgcm documentation: A flag indicating how to handle boundaries: * None: Do not apply any boundary conditions. Raise an error if

boundary conditions are required for the operation.

- 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
- 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **attrs** (*dict, optional*) – Dictionary of attributes to add to resultant arrays. Requires that q is DataArray. For example: *attrs={'name': 'varname', 'long_name': 'longvarname', 'units': 'units'}*

Returns

- *DataArray of dqdxi, the gradient of q in the xi-direction with*
- *attributes altered to reflect calculation.*

Notes

dqdxi = dqdx*dzdz - dqdz*dzdx

Derivatives are taken in the ROMS curvilinear grid native xi-direction.

These derivatives properly account for the fact that ROMS vertical coordinates are s coordinates and therefore can vary in time and space.

This will alter the number of points in the xi and s dimensions.

Examples

```
>>> ds.salt.xroms.ddxi(xgrid)
```

ddz(*xgrid*, *hcoord=None*, *scoord=None*, *hboundary='extend'*, *hfill_value=None*, *sboundary='extend'*, *sfill_value=None*, *attrs=None*)

Calculate d/dz for a variable.

Parameters

- **xgrid** – xgcm grid
- **hcoord** (*string, optional.*) – Name of horizontal grid to interpolate output to. Options are 'rho', 'psi', 'u', 'v'.
- **scoord** (*string, optional.*) – Name of vertical grid to interpolate output to. Options are 's_rho', 's_w', 'rho', 'w'.
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for grid changes. From xgcm documentation: A flag indicating how to handle boundaries: * None: Do not apply any boundary conditions. Raise an error if

boundary conditions are required for the operation.

  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for calculating z derivative. This same value will be used for grid changes too. From xgcm documentation: A flag indicating how to handle boundaries: * None: Do not apply any boundary conditions. Raise an error if

  boundary conditions are required for the operation.

  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary fill value associated with sboundary input. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **attrs** (*dict, optional*) – Dictionary of attributes to add to resultant arrays. Requires that q is DataArray. For example: *attrs={'name': 'varname', 'long_name': 'longvarname', 'units': 'units'}*

Returns

- *DataArray of vertical derivative of variable with*
- *attributes altered to reflect calculation.*

Notes

This will alter the number of points in the s dimension.

Examples

```
>>> ds.salt.xroms.ddz(xgrid)
```

gridmean(*xgrid*, *dim*)

Calculate mean accounting for variable spatial grid.

Parameters

- **xgrid** – xgcm grid
- **dim** (*str, list, tuple*) – Spatial dimension names to average over. In the *xgcm* convention, the allowable names are 'Z', 'Y', or 'X'.

Returns

- *DataArray or ndarray of average calculated over dim accounting*

- *for variable spatial grid.*

Notes

If result is DataArray, long name attribute is modified to describe calculation.

Examples

Note that the following two approaches are equivalent: >>> app1 = ds.u.xroms.gridmean(xgrid, ('Y','X')) >>> app2 = (ds.u*ds.dy_u*ds.dx_u).sum(('eta_rho','xi_u'))/(ds.dy_u*ds.dx_u).sum(('eta_rho','xi_u')) >>> np.allclose(app1, app2)

gridsum(*xgrid*, *dim*)

Calculate sum accounting for variable spatial grid.

Parameters

- **xgrid** – xgcm grid
- **dim** (*str, list, tuple*) – Spatial dimension names to sum over. In the *xgcm* convention, the allowable names are 'Z', 'Y', or 'X'.

Returns

- *DataArray or ndarray of sum calculated over dim accounting*
- *for variable spatial grid.*

Notes

If result is DataArray, long name attribute is modified to describe calculation.

Examples

Note that the following two approaches are equivalent: >>> app1 = ds.u.xroms.gridsum(xgrid, ('Z','X')) >>> app2 = (ds.u*ds.dz_u * ds.dx_u).sum(('s_rho','xi_u')) >>> np.allclose(app1, app2)

interpll(*lons*, *lats*, *which='pairs'*, ***kwargs*)

Interpolate var to lons/lats positions.

Wraps xESMF to perform proper horizontal interpolation on non-flat Earth.

Parameters

- **lons** (*list, ndarray*) – Longitudes to interpolate to. Will be flattened upon input.
- **lats** (*list, ndarray*) – Latitudes to interpolate to. Will be flattened upon input.
- **which** (*str, optional*) – Which type of interpolation to do: * "pairs": lons/lats as unstructured coordinate pairs

  (in xESMF language, LocStream).

  – "grid": 2D array of points with 1 dimension the lons and the other dimension the lats.

- ****kwargs** – passed on to xESMF Regridder class

Returns

- *DataArray of var interpolated to lons/lats. Dimensionality will be the*
- *same as var except the Y and X dimensions will be 1 dimension called*
- *"locations" that lons.size if which=='pairs', or 2 dimensions called*
- *"lat" and "lon" if which=='grid' that are of lats.size and lons.size,*
- *respectively.*

Notes

var cannot have chunks in the Y or X dimensions.

cf-xarray should still be usable after calling this function.

Examples

To return 1D pairs of points, in this case 3 points: >>> xroms.interpll(var, [-96, -97, -96.5], [26.5, 27, 26.5], which='pairs') To return 2D pairs of points, in this case a 3x3 array of points: >>> xroms.interpll(var, [-96, -97, -96.5], [26.5, 27, 26.5], which='grid')

order()

Reorder self to typical dimensional ordering.

Returns

- *DataArray with dimensional order ['T', 'Z', 'Y', 'X'], or whatever subset of*
- *dimensions are present in var.*

Notes

Do not consider previously-selected dimensions that are kept on as coordinates but cannot be transposed anymore. This is accomplished with *.reset_coords(drop=True)*.

Examples

```
>>> ds.temp.xroms.order()
```

sel2d(*lon0*, *lat0*)

Find the value of the var at closest location to lon0,lat0.

Parameters

- **lon0** (*float, int*) – Longitude of comparison point.
- **lat0** (*float, int*) – Latitude of comparison point.

Return type

DataArray value(s) of closest location to lon0/lat0.

Notes

This function uses Great Circle distance to calculate distances assuming longitudes and latitudes as point coordinates. Uses cartopy function *Geodesic*: https://scitools.org.uk/cartopy/docs/latest/cartopy/geodesic.html

This wraps *argsel2d*.

Examples

```
>>> ds.temp.xroms.sel2d(-96, 27)
```

to_grid(*xgrid*, *hcoord=None*, *scoord=None*, *hboundary='extend'*, *hfill_value=None*, *sboundary='extend'*, *sfill_value=None*)

Implement grid changes.

Parameters

- **xgrid** – xgcm grid
- **hcoord** (*string, optional.*) – Name of horizontal grid to interpolate output to. Options are 'rho', 'psi', 'u', 'v'.
- **scoord** (*string, optional.*) – Name of vertical grid to interpolate output to. Options are 's_rho', 's_w', 'rho', 'w'.
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for grid changes. From xgcm documentation: A flag indicating how to handle boundaries: * None: Do not apply any boundary conditions. Raise an error if

  boundary conditions are required for the operation.

  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for grid changes. From xgcm documentation: A flag indicating how to handle boundaries: * None: Do not apply any boundary conditions. Raise an error if

  boundary conditions are required for the operation.

  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.
- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

Returns

- *DataArray interpolated onto hcoord horizontal and scoord*
- *vertical grids.*

Notes

If var is already on selected grid, nothing happens.

Examples

```
>>> ds.salt.xroms.to_grid(xgrid, hcoord='rho', scoord='w')
```

zslice(*xgrid*, *depths*, *z=None*)

Interpolate var to depths.

This wraps *xgcm transform* function for slice interpolation, though *transform* has additional functionality. See `xroms.isoslice` for full docs.

Parameters

- **xgrid** – xgcm grid
- **depths** (*list, ndarray*) – Values to interpolate to (called iso_values in other functions). Should be negative if below mean sea level. If input as array, should be 1D.
- **z** (*DataArray, optional*) – Array that var is interpolated onto (e.g., z coordinates or density). The "vertical" coordinate is selected by default. Use this option if you want to interpolate with z depths constant in time and input the appropriate z coordinate (e.g. z_rho0).

Returns

- *DataArray of var interpolated to depths. Dimensionality will be the*
- *same as var except with dim dimension of size of depths.*

Notes

var cannot have chunks in the dimension dim.

cf-xarray should still be usable after calling this function.

Examples

To calculate temperature onto fixed depths:

```
>>> ds.temp.xroms.zslice(depths)
```

To calculate temperature onto fixed depths without considering time for z coord:

```
>>> ds.temp.xroms.zslice(depths, z=ds.temp.z_rho0)
```

class xroms.accessor.**xromsDatasetAccessor**(*ds*)

Bases: `object`

Accessor for Datasets.

Attributes

EKE
Calculate EKE [m^2/s^2], on rho grid.

KE
Calculate kinetic energy [kg/(m*s^2)], on rho/rho grids.

M2
Calculate the horizontal buoyancy gradient on rho/w grids.

N2
Calculate buoyancy frequency squared on rho/w grids.

buoyancy
Calculate buoyancy on rho/rho grids.

convergence
Calculate convergence, rho/rho grid.

convergence_norm
Calculate normalized surface convergence, rho/rho grid.

dudz
Calculate dudz [1/s] on u/w grids.

dvdz
Calculate dvdz [1/s] on v/w grids.

east
Rotate grid-aligned u velocity to be eastward.

ertel
Calculate Ertel potential vorticity of buoyancy on rho/rho grids.

north
Rotate grid-aligned v velocity to be northward.

omega
Calculate s-grid vertical velocity on [horizontal]/[vertical] grids.

rho
Return existing rho or calculate, on rho/rho grids.

sig0
Calculate potential density referenced to z=0, on rho/rho grids.

speed
Calculate horizontal speed [m/s] from u and v components, on rho/rho grids.

u
Rotate eastward velocity to be grid-aligned u velocity

ug
Calculate geostrophic u velocity from zeta, on u grid.

v
Rotate northward velocity to be grid-aligned v velocity

vertical_shear
Calculate vertical shear [1/s], rho/w grids.

vg
Calculate geostrophic v velocity from zeta, on v grid.

vort
Calculate vertical relative vorticity, psi/w grids.

w
Calculate vertical velocity on [horizontal]/[vertical] grids.

xgrid

Methods

ddeta(varname[, hcoord, scoord, hboundary, ...])	Calculate d/deta for a variable.
ddxi(varname[, hcoord, scoord, hboundary, ...])	Calculate d/dxi for a variable.
ddz(varname[, hcoord, scoord, hboundary, ...])	Calculate d/dz for a variable.
east_rotated(angle[, name])	Rotate eastward velocity by angle.
mld([thresh])	Calculate mixed layer depth [m] on rho grid.
north_rotated(angle[, name])	Rotate northward velocity by angle.
set_grid(xgrid)	If you already have a xgrid object and don't want to rerun
subset([X, Y])	Subset model output horizontally using isel, properly accounting for horizontal grids.
to_grid(varname[, hcoord, scoord, ...])	Implement grid changes.
zslice(varname, depths[, z])	Interpolate var to depths.

find_horizontal_velocities

property EKE

Calculate EKE [m^2/s^2], on rho grid.

Notes

EKE = 0.5*(ug^2 + vg^2) Puts geostrophic speed on rho grid.

See *xroms.EKE* for full docstring.

Examples

```
>>> ds.xroms.EKE
```

property KE

Calculate kinetic energy [kg/(m*s^2)], on rho/rho grids.

Notes

Uses speed that has been extended out to the rho grid and rho0.

See *xroms.KE* for full docstring.

Examples

```
>>> ds.xroms.KE
```

property M2

Calculate the horizontal buoyancy gradient on rho/w grids.

Notes

See *xroms.M2* for full docstring.

hboundary set to 'extend' and *sboundary='fill'* with *sfill_value=np.nan*.

Examples

```
>>> ds.xroms.M2
```

property N2

Calculate buoyancy frequency squared on rho/w grids.

Notes

See *xroms.N2* for full docstring.

sboundary set to 'fill' with *sfill_value=np.nan*.

Examples

```
>>> ds.xroms.N2
```

_eastnorth2uv()

Call the velocity rotation for accessor.

_eastnorth_rotated(*angle*, *include_vars_adcp=False*, ***kwargs*)

Call the velocity rotation for accessor.

include_vars_adcp
: [bool] If True, include all variables that might be compared with ADCP data and ways to convert between: east_rotated, north_rotated, angle, east, north, grid_angle.

_uv2eastnorth()

Call the velocity rotation for accessor.

property buoyancy

Calculate buoyancy on rho/rho grids.

Notes

See *xroms.buoyancy* for full docstring.

Examples

```
>>> ds.xroms.buoyancy
```

property convergence

Calculate convergence, rho/rho grid.

Notes

See *xroms.convergence* for full docstring.

hboundary and *sboundary* both set to 'extend'.

Examples

```
>>> ds.xroms.convergence
```

property convergence_norm

Calculate normalized surface convergence, rho/rho grid.

The surface currents are selected for this calculation, so return is *[T,Y,X]*. The convergence is normalized by f. It is dimensionless.

Notes

See *xroms.convergence* for full docstring.

hboundary and *sboundary* both set to 'extend'.

Examples

```
>>> ds.xroms.convergence_norm
```

ddeta(*varname*, *hcoord=None*, *scoord=None*, *hboundary='extend'*, *hfill_value=None*, *sboundary='extend'*, *sfill_value=None*, *attrs=None*)

Calculate d/deta for a variable.

Parameters

- **varname** (*str*) – Name of variable in Dataset to operate on.
- **hcoord** (*string, optional.*) – Name of horizontal grid to interpolate output to. Options are 'rho', 'psi', 'u', 'v'.
- **scoord** (*string, optional.*) – Name of vertical grid to interpolate output to. Options are 's_rho', 's_w', 'rho', 'w'.

- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for calculating horizontal derivative of var. This same value will be used for grid changes too. From xgcm documentation: A flag indicating how to handle boundaries: * None: Do not apply any boundary conditions. Raise an error if

  boundary conditions are required for the operation.

  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for calculating horizontal derivative of var. This same value will be used for grid changes too. From xgcm documentation: A flag indicating how to handle boundaries: * None: Do not apply any boundary conditions. Raise an error if

  boundary conditions are required for the operation.

  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **attrs** (*dict, optional*) – Dictionary of attributes to add to resultant arrays. Requires that q is DataArray. For example: *attrs={'name': 'varname', 'long_name': 'longvarname', 'units': 'units'}*

Returns

- *DataArray of dqdeta, the gradient of q in the eta-direction with*
- *attributes altered to reflect calculation.*

Notes

dqdeta = dqdy*dzdz - dqdz*dzdy

Derivatives are taken in the ROMS curvilinear grid native eta-direction.

These derivatives properly account for the fact that ROMS vertical coordinates are s coordinates and therefore can vary in time and space.

This will alter the number of points in the eta and s dimensions.

Examples

```
>>> ds.xroms.ddeta('salt')
```

ddxi(*varname*, *hcoord=None*, *scoord=None*, *hboundary='extend'*, *hfill_value=None*, *sboundary='extend'*, *sfill_value=None*, *attrs=None*)

Calculate d/dxi for a variable.

Parameters

- **varname** (*str*) – Name of variable in Dataset to operate on.
- **hcoord** (*string, optional.*) – Name of horizontal grid to interpolate output to. Options are 'rho', 'psi', 'u', 'v'.
- **scoord** (*string, optional.*) – Name of vertical grid to interpolate output to. Options are 's_rho', 's_w', 'rho', 'w'.
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for calculating horizontal derivative of var. This same value will be used for all horizontal grid changes too. From xgcm documentation: A flag indicating how to handle boundaries: * None: Do not apply any boundary conditions. Raise an error if

  boundary conditions are required for the operation.

  – 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)

  – 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for calculating horizontal derivative of var. This same value will be used for all vertical grid changes too. From xgcm documentation: A flag indicating how to handle boundaries: * None: Do not apply any boundary conditions. Raise an error if

  boundary conditions are required for the operation.

  – 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)

  – 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **attrs** (*dict, optional*) – Dictionary of attributes to add to resultant arrays. Requires that q is DataArray. For example: *attrs={'name': 'varname', 'long_name': 'longvarname', 'units': 'units'}*

Returns

- *DataArray of dqdxi, the gradient of q in the xi-direction with*

- *attributes altered to reflect calculation.*

Notes

dqdxi = dqdx*dzdz - dqdz*dzdx

Derivatives are taken in the ROMS curvilinear grid native xi-direction.

These derivatives properly account for the fact that ROMS vertical coordinates are s coordinates and therefore can vary in time and space.

This will alter the number of points in the xi and s dimensions.

Examples

```
>>> ds.xroms.ddxi('salt')
```

ddz(*varname*, *hcoord=None*, *scoord=None*, *hboundary='extend'*, *hfill_value=None*, *sboundary='extend'*, *sfill_value=None*, *attrs=None*)

Calculate d/dz for a variable.

Parameters

- **varname** (*str*) – Name of variable in Dataset to operate on.
- **hcoord** (*string, optional.*) – Name of horizontal grid to interpolate output to. Options are 'rho', 'psi', 'u', 'v'.
- **scoord** (*string, optional.*) – Name of vertical grid to interpolate output to. Options are 's_rho', 's_w', 'rho', 'w'.
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for grid changes. From xgcm documentation: A flag indicating how to handle boundaries: * None: Do not apply any boundary conditions. Raise an error if

  boundary conditions are required for the operation.

  – 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)

  – 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for calculating z derivative. This same value will be used for grid changes too. From xgcm documentation: A flag indicating how to handle boundaries: * None: Do not apply any boundary conditions. Raise an error if

  boundary conditions are required for the operation.

  – 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)

– 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary fill value associated with sboundary input. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **attrs** (*dict, optional*) – Dictionary of attributes to add to resultant arrays. Requires that q is DataArray. For example: *attrs={'name': 'varname', 'long_name': 'longvarname', 'units': 'units'}*

Returns

- *DataArray of vertical derivative of variable with*
- *attributes altered to reflect calculation.*

Notes

This will alter the number of points in the s dimension.

Examples

```
>>> ds.xroms.ddz('salt')
```

property dudz

Calculate dudz [1/s] on u/w grids.

Notes

See *xroms.dudz* for full docstring.

sboundary is set to 'extend'.

Examples

```
>>> ds.xroms.dudz
```

property dvdz

Calculate dvdz [1/s] on v/w grids.

Notes

See *xroms.dvdz* for full docstring.

sboundary is set to 'extend'.

Examples

```
>>> ds.xroms.dvdz
```

property east

Rotate grid-aligned u velocity to be eastward.

Notes

See *xroms.rotate_vectors* for full docstring.

Examples

```
>>> ds.xroms.east
```

east_rotated(*angle*, *name=None*, ***kwargs*)

Rotate eastward velocity by angle.

Parameters

- **angle** (*float, xr.DataArray*) – Angle to rotate eastward, northward velocities by to get x component of rotated velocities.
- **name** (*str, optional*) – If input, will be used for output array name.
- **kwargs** (*optional*) – will be input to `xroms.rotate_vectors()`.

Notes

See *xroms.rotate_vectors()* for full docstring.

Examples

```
>>> ds.xroms.east_rotated(angle, reference="compass", isradians=False, name=
↪"along_channel")
```

property ertel

Calculate Ertel potential vorticity of buoyancy on rho/rho grids.

Notes

See *xroms.ertel* for full docstring.

hboundary and *sboundary* both set to 'extend'.

Examples

```
>>> ds.xroms.ertel
```

find_horizontal_velocities()

mld(*thresh=0.03*)

Calculate mixed layer depth [m] on rho grid.

property north

Rotate grid-aligned v velocity to be northward.

Notes

See *xroms.rotate_vectors* for full docstring.

Examples

```
>>> ds.xroms.north
```

north_rotated(*angle*, *name=None*, ***kwargs*)

Rotate northward velocity by angle.

Parameters

- **angle** (*float, xr.DataArray*) – Angle to rotate eastward, northward velocities by to get y component of rotated velocities.
- **name** (*str, optional*) – If input, will be used for output array name.
- **kwargs** (*optional*) – will be input to `xroms.rotate_vectors()`.

Notes

See *xroms.rotate_vectors()* for full docstring.

Examples

```
>>> ds.xroms.north_rotated(angle, reference="compass", isradians=False, name=
↪"across_channel")
```

property omega

Calculate s-grid vertical velocity on [horizontal]/[vertical] grids.

Notes

See *xroms.omega* for full docstring.

Examples

```
>>> ds.xroms.omega
```

property rho

Return existing rho or calculate, on rho/rho grids.

Notes

See *xroms.density* for full docstring.

Examples

```
>>> ds.xroms.rho
```

set_grid(*xgrid*)

If you already have a xgrid object and don't want to rerun

Or, you want to have more options in the xgrid setup, input it to the xroms accessor this way.

Examples

```
>>> ds.xroms.set_grid(xgrid)
```

property sig0

Calculate potential density referenced to z=0, on rho/rho grids.

Notes

See *xroms.potential_density* for full docstring.

Examples

```
>>> ds.xroms.sig0
```

property speed

Calculate horizontal speed [m/s] from u and v components, on rho/rho grids.

Notes

speed = np.sqrt(u^2 + v^2)

Uses 'extend' for horizontal boundary.

See *xroms.speed* for full docstring.

Examples

```
>>> ds.xroms.speed
```

subset(*X=None*, *Y=None*)

Subset model output horizontally using isel, properly accounting for horizontal grids.

Parameters

- **X** (*slice, optional*) – Slice in X dimension using form *X=slice(start, stop, step)*. For example, >>> X=slice(20,40,2) Indices are used for rho grid, and psi grid is reduced accordingly.
- **Y** (*slice, optional*) – Slice in Y dimension using form *Y=slice(start, stop, step)*. For example, >>> Y=slice(20,40,2) Indices are used for rho grid, and psi grid is reduced accordingly.

Returns

- *Dataset with form as if model had been run at the subsetted size. That is, the outermost*
- *cells of the rho grid are like ghost cells and the psi grid is one inward from this size*
- *in each direction.*

Notes

X and Y must be slices, not single numbers.

Examples

Subset only in Y direction: >>> ds.xroms.subset(Y=slice(50,100)) Subset in X and Y: >>> ds.xroms.subset(X=slice(20,40), Y=slice(50,100))

to_grid(*varname*, *hcoord=None*, *scoord=None*, *hboundary='extend'*, *hfill_value=None*, *sboundary='extend'*, *sfill_value=None*)

Implement grid changes.

Parameters

- **varname** (*str*) – Name of variable in Dataset to operate on.
- **hcoord** (*string, optional.*) – Name of horizontal grid to interpolate output to. Options are 'rho', 'psi', 'u', 'v'.
- **scoord** (*string, optional.*) – Name of vertical grid to interpolate output to. Options are 's_rho', 's_w', 'rho', 'w'.
- **hboundary** (*string, optional*) – Passed to *grid* method calls; horizontal boundary selection for grid changes. From xgcm documentation: A flag indicating how to handle boundaries: * None: Do not apply any boundary conditions. Raise an error if

boundary conditions are required for the operation.

  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **hfill_value** (*float, optional*) – Passed to *grid* method calls; horizontal boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.
- **sboundary** (*string, optional*) – Passed to *grid* method calls; vertical boundary selection for grid changes. From xgcm documentation: A flag indicating how to handle boundaries: * None: Do not apply any boundary conditions. Raise an error if

  boundary conditions are required for the operation.

  - 'fill': Set values outside the array boundary to fill_value (i.e. a Neumann boundary condition.)
  - 'extend': Set values outside the array to the nearest array value. (i.e. a limited form of Dirichlet boundary condition.

- **sfill_value** (*float, optional*) – Passed to *grid* method calls; vertical boundary selection fill value. From xgcm documentation: The value to use in the boundary condition with *boundary='fill'*.

Returns

- *DataArray interpolated onto hcoord horizontal and scoord*
- *vertical grids.*

Notes

If var is already on selected grid, nothing happens.

Examples

```
>>> ds.xroms.to_grid('salt', hcoord='rho', scoord='w')
```

property u

Rotate eastward velocity to be grid-aligned u velocity

Notes

See *xroms.rotate_vectors* for full docstring.

Examples

```
>>> ds.xroms.u
```

property ug

Calculate geostrophic u velocity from zeta, on u grid.

Notes

ug = -g * zeta_xi / (d xi * f) # on u grid

See *xroms.uv_geostrophic* for full docstring.

Examples

```
>>> ds.xroms.ug
```

property v

Rotate northward velocity to be grid-aligned v velocity

Notes

See *xroms.rotate_vectors* for full docstring.

Examples

```
>>> ds.xroms.v
```

property vertical_shear

Calculate vertical shear [1/s], rho/w grids.

Notes

See *xroms.vertical_shear* for full docstring.

hboundary is set to 'extend'.

Examples

```
>>> ds.xroms.vertical_shear
```

property vg

Calculate geostrophic v velocity from zeta, on v grid.

Notes

vg = g * zeta_eta / (d eta * f) # on v grid

See *xroms.uv_geostrophic* for full docstring.

Examples

```
>>> ds.xroms.vg
```

property vort

Calculate vertical relative vorticity, psi/w grids.

Notes

See *xroms.relative_vorticity* for full docstring.

hboundary and *sboundary* both set to 'extend'.

Examples

```
>>> ds.xroms.vort
```

property w

Calculate vertical velocity on [horizontal]/[vertical] grids.

Notes

See *xroms.w* for full docstring.

Examples

```
>>> ds.xroms.w
```

property xgrid

zslice(*varname*, *depths*, *z=None*)

Interpolate var to depths.

This wraps *xgcm transform* function for slice interpolation, though *transform* has additional functionality. See `xroms.isoslice` for full docs.

Parameters

- **depths** (*list, ndarray*) – Values to interpolate to (called iso_values in other functions). Should be negative if below mean sea level. If input as array, should be 1D.
- **z** (*DataArray, optional*) – Array that var is interpolated onto (e.g., z coordinates or density). The "vertical" coordinate is selected by default. Use this option if you want to interpolate with z depths constant in time and input the appropriate z coordinate (e.g. z_rho0).

Returns

- *DataArray of var interpolated to depths. Dimensionality will be the*
- *same as var except with dim dimension of size of depths.*

Notes

var cannot have chunks in the dimension dim.

cf-xarray should still be usable after calling this function.

Examples

To calculate temperature onto fixed depths:

```
>>> ds.temp.xroms.zslice(depths)
```

To calculate temperature onto fixed depths without considering time for z coord:

```
>>> ds.temp.xroms.zslice(depths, z=ds.temp.z_rho0)
```

1.7 What's New

1.7.1 v0.6.0 (February 9, 2024)

- fixed error in `derived.py`'s `uv_geostrophic` function after being pointed out by @ak11283
- updated docs so mostly well-formatted and working

1.7.2 v0.5.3 (October 11, 2023)

- change to `roms_dataset()` so that input flag `include_3D_metrics` also controls if `ds["3d"] = True`.

1.7.3 v0.5.2 (October 4, 2023)

- small fix to `roms_dataset()` processing to enable running it twice

1.7.4 v0.5.1 (September 14, 2023)

- renamed all references to "divergence" to "convergence" instead

1.7.5 v0.5.0 (September 12, 2023)

- the mixed layer depth function now returns positive values

1.7.6 v0.4.7 (September 8, 2023)

- Fixed attributes for accessor method `div_norm`

1.7.7 v0.4.6 (July 31, 2023)

- fixed `ds.xroms.div` and `ds.xroms.div_norm` in the case that `u` and `v` need to be calculated from other velocities like `east` and `north`.

1.7.8 v0.4.5 (July 27, 2023)

- typo fix

1.7.9 v0.4.4 (July 27, 2023)

- added accessor function `find_horizontal_velocities()` which returns the names of the horizontal velocities since they sometimes have different names, but still there are only a few options.

1.7.10 v0.4.3 (July 27, 2023)

- zkey is checked for but not required in interpll now

1.7.11 v0.4.2 (July 27, 2023)

- changes to roms_dataset
  - If "coordinates" are found in attrs for a variable, they are moved to "encoding" now because everything works better then.
  - Recreated the zslice function in the accessor for both Dataset and DataArray (instead of just using the default isoslice).
  - updated docs and tests accordingly.

1.7.12 v0.4.1 (July 27, 2023)

- can now pass kwargs to xe.Regridder in interpll

1.7.13 v0.4.0 (July 25, 2023)

- hopefully fixing issue reordering dimensions when extra coords present
- divergence calculation was added to derived.py
- accessor changes:
  - xgrid is run automatically when accessor is used, which could be too slow for some uses
  - div and div_norm properties added to accessor
  - div_norm is the surface divergence normalized by f
- added tests for new functions
- hopefully fixed build issue on several OSes by pinning `h5py < 3.2`, see for reference https://github.com/h5py/h5py/issues/1880, https://github.com/conda-forge/h5py-feedstock/issues/92
- updated docs

1.7.14 v0.3.3 (July 11, 2023)

- do not use Z coords if 2d

1.7.15 v0.3.2 (June 23, 2023)

- More fixes to the rotation accessor options

1.7.16 v0.3.1 (June 22, 2023)

- made east/north variable names have two options

1.7.17 v0.3.0 (June 12, 2023)

- can rotate along-grid velocities to be eastward and northward
- can also rotate to be along an arbitrary angle (to be along-channel for example)

1.7.18 v0.2.3 (May 24, 2023)

- updating versioning approach
- the xgcm grid is no longer attached to every variable in a Dataset. Because of this:
  - Several `xroms` accessor functions now require the grid to be input
  - Additionally because of the grid not being available, the Dataset is no longer available within the DataArray accessor, making it so that functions that change the grid size in any dimension no longer know about other coordinates to use. Therefore, these `xroms` accessor functions for DataArrays no longer work (e.g. ddeta, ddxi, etc). All `xroms` Dataset accessor functions still work, and the grid object is still saved to the Dataset `xroms` accessor.
- You can set up the grid object directly in your `xroms` Dataset accessor with `ds.xroms.set_grid(grid)`, otherwise it will be calculated internally.

- `xroms` functions for opening model output files will be deprecated in the future; use `xarray` functions directly instead of opening model output through `xroms`, and then run `xroms.roms_dataset()` to add functionality to your Dataset and to calculate your `xgcm` grid object.
- tests have been updated
- `xroms` works with newest version of `xgcm`
- changed all references to the `xgcm` grid to `xgrid` since there is now a "grid" attribute in some Datasets.
- updated example notebooks to be formal docs
- added a ROMS example dataset, available with `xroms.datasets.fetch_ROMS_example_full_grid()`.

PYTHON MODULE INDEX

X

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

M

N

O

P

R

S

T

U

V

W

X

Z