Easier, Better, Faster, Shorter: Updates to grid-aware analysis with xgcm

•••

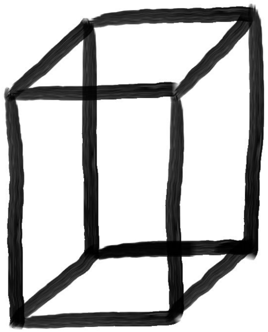
Dianne Deauna | University of Hawai'i at Mānoa | SIParCS intern | Project 8
Mentors: Anderson Banihirwe (NCAR), Julius Busecke (Columbia University), and Deepak Cherian (CGD)

Land Acknowledgement

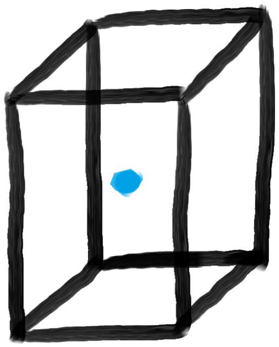
I would like to acknowledge the ‘āina on which I am coming from you today, from the ‘ili āina of Kauwalaa, the ahupua’a of Mānoa, in the moku of Kona, on the mokupuni of O’ahu, in the pae’āina of Hawai’i. I recognize that her majesty Queen Lili’uokalani yielded the Hawaiian Kingdom and these territories under duress and protest to the United States to avoid the bloodshed of her people, and that Hawai’i remains an illegally occupied state of America. I further recognize that generations of Indigenous Hawaiians and their knowledge systems shaped Hawai’i in sustainable ways that allow me to enjoy these gifts today. For this I am grateful as a guest, and I seek to support the varied strategies that the Indigenous peoples of Hawai’i are using to protect their land and communities.

Adapted from: http://manoa.hawaii.edu/nhpol/language-option/pathways/auamo/

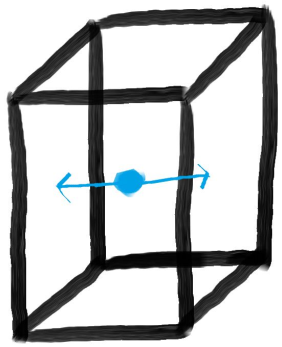
Ocean modeling is mostly representing the ocean as a lot of rectangular cubes



Allows for efficient integration of PDEs forwards through time

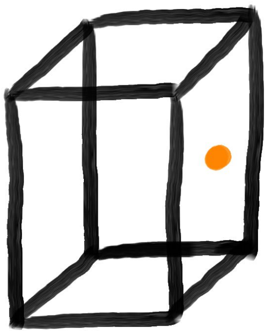


Scalar quantity: temperature
Location: center

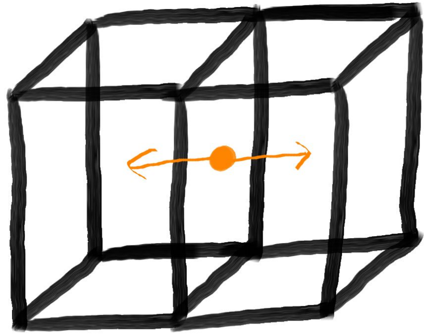


Calculate average temperature along the x-axis: need distance from the center to the cell faces

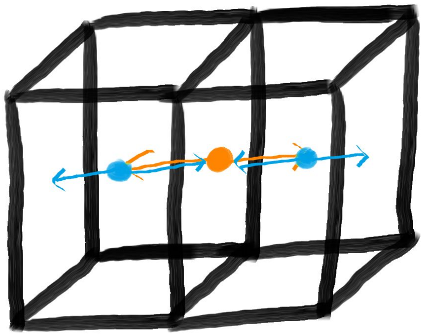
Assigning velocity values to shifted locations within a grid cell makes calculations numerically efficient



Vector quantity: u-velocity
Location: “eastern” face
(shifted to the right
relative to temperature)

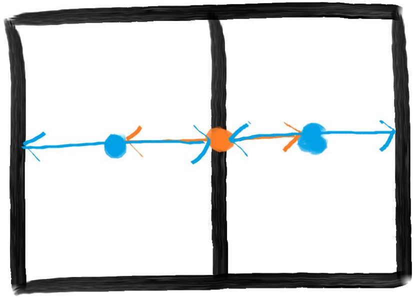


Calculate average u-velocity along the x-axis: need distance from the cell face to the centers

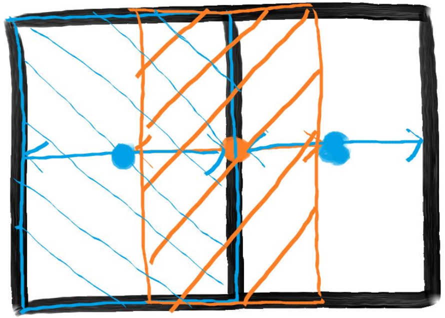


Postprocessing ocean models require tools that can keep track of these distances for *grid-aware operations*

In addition to distances, postprocessing tools also need to keep track of complex cell geometries



Consider: Area along X,Y axis



Temperature and u-velocity areas are shifted in position and not necessarily equal to each other xarray

Python package

- Makes working with n-dimensional arrays (often provided as netcdf files) more efficient
- Labels raw arrays with dimensions, coordinates, and attributes

GCM

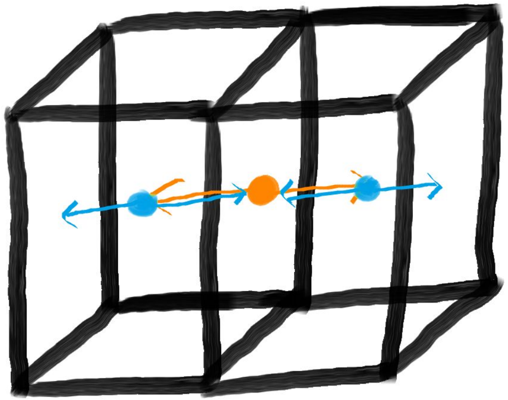
Python package

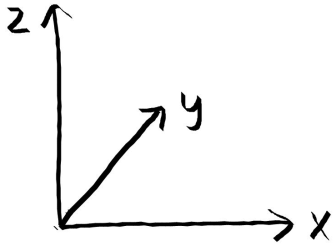
- General Circulation Model postprocessing with xarray
- Has sophisticated metric handling for staggered grid datasets
- Has built-in grid-aware operations such as average, integrate, etc.

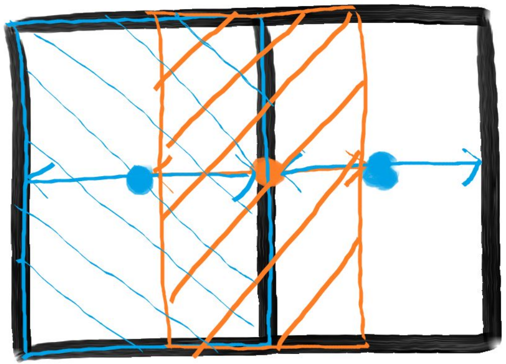
metrics

definition

- Information about grid cell geometry in physical space
- Includes:
  - Distance along ‘X’, ‘Y’, or ‘Z’ axis,
  - Areas along (‘X’,’Y’), (‘Y’,’Z’), and (‘X’,’Z’),
  - Volume along (‘X’,’Y’,’Z’)
- Usually not explicitly defined in model outputs for all variables at all positions → there is a need for interpolation







Updated xgcm's metric handling with three new methods

set_metrics()	• Enables overwriting of previously assigned metrics and allows for assigning multiple ones on the same axis but with different dimensions
interp_like()	• Allows for the interpolation of a data array to the positions of another data array
get_metric()	• Selects for the required metric for a data variable along a specified axis for grid-aware operations • Incorporates *interp_like()* to allow for the automatic interpolation of missing metrics from available metric values on surrounding positions

Interactive Jupyter notebook: bit.ly/xgcm_demo_siparcs2021

Interactive Jupyter notebook:
bit.ly/xgcm_demo_siparcs2021

Load data from an Earth System Model

```
ds_subset
```

xarray.Dataset

▶ Dimensions: (**lev**: 75, **time**: 1980, x: 20, x_c: 20, y: 38)

▼ Coordinates:

lat	(y, x)	float64	dask.array<chunksize=(38, 20), meta=np.ndarra...
lev	(lev)	float64	0.5058 1.556 ... 5.902e+03
lon	(y, x)	float64	dask.array<chunksize=(38, 20), meta=np.ndarra...
time	(time)	object	1850-01-16 12:00:00 ... 2014-12-...
areacello	(y, x)	float32	dask.array<chunksize=(38, 20), meta=np.ndarra...
lat_u	(y, x_c)	float64	dask.array<chunksize=(38, 20), meta=np.ndarra...
lon_u	(y, x_c)	float64	dask.array<chunksize=(38, 20), meta=np.ndarra...

▼ Data variables:

thetao	(time, lev, y, x)	float32	dask.array<chunksize=(4, 75, 38, 20), meta=np....
uo	(time, lev, y, x_c)	float32	dask.array<chunksize=(3, 75, 38, 20), meta=np....

Create a grid object using xgcm which contains all information

```
from xgcm import Grid
grid = Grid(
    ds_subset,
    coords={
        'X':{'center':'x', 'right':'x_c'},
        'Y':{'center':'y', 'right':'y_c'},
        'Z':{'center':'lev'},
    },
    periodic=False,
    boundary='extend',
    metrics={('X','Y'): 'areacello'}
)
grid._metrics
```

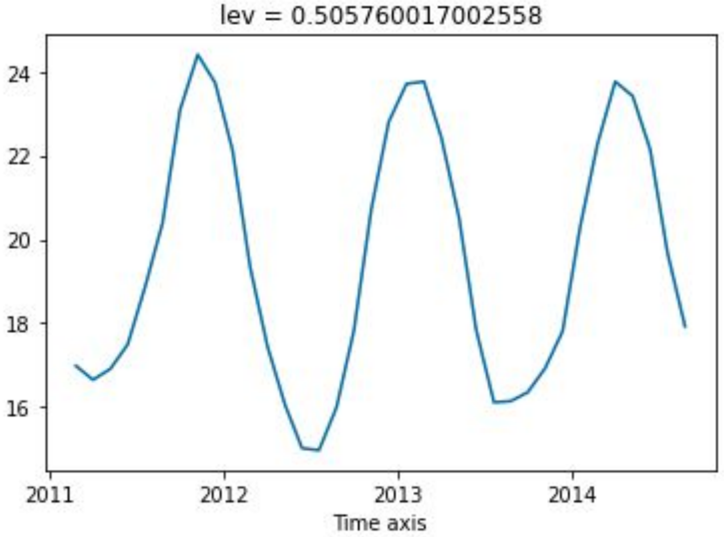
```
{frozenset({'X',
            'Y'}): [<xarray.DataArray 'areacello' (y: 38, x: 20)>
  dask.array<getitem, shape=(38, 20), dtype=float32, chunksize=(38, 20), chunktype=numpy.ndarray>
  Dimensions without coordinates: y, x
  Attributes:
      cell_methods:      area: sum
      description:       Cell areas for any grid used to report ocean variables...
      history:           none
      long_name:         Grid-Cell Area
      online_operation:  once
      standard_name:     cell_area
      units:             m2]}
```

Interactive Jupyter notebook:
bit.ly/xgcm_demo_siparcs2021

Calculating area-weighted temperature is straightforward...

```
mean_sst = grid.average(sst,['X','Y'])
mean_sst.plot()
```

```
[<matplotlib.lines.Line2D at 0x7fce59849460>]
```



Interactive Jupyter notebook:
bit.ly/xgcm_demo_siparcs2021

...but not for area-weighted u-velocity (old version of xgcm)

```
import xgcm
xgcm.__version__
```

```
'0.5.1'
```

```
mean_uo = grid.average(uo,['X','Y'])
mean_uo.plot()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-16-7b3cf12a0d9a> in <module>
----> 1 mean_uo = grid.average(uo,['X','Y'])
      2 mean_uo.plot()

/srv/conda/envs/notebook/lib/python3.8/site-packages/xgcm/grid.py in average(self, da, axis, **kwargs)
   1734             The averaged data
   1735         """
-> 1736         weight = self.get_metric(da, axis)
   1737         weighted = da.weighted(weight)
   1738

/srv/conda/envs/notebook/lib/python3.8/site-packages/xgcm/grid.py in get_metric(self, array, axes)
   1339                     pass
   1340         if metric_vars is None:
-> 1341             raise KeyError(
   1342                 "Unable to find any combinations of metrics for "
   1343                 "array dims %r and axes %r" % (array_dims, axes)

KeyError: "Unable to find any combinations of metrics for array dims {'x_c', 'time', 'y'} and axes ['X', 'Y']"
```

Old way = lengthy code :(

Interactive Jupyter notebook:
bit.ly/xgcm_demo_siparcs2021

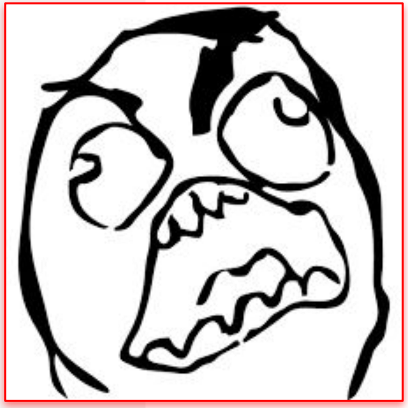
```
from xgcm import Grid

# Step 1: Create a grid object with the available metric
grid = Grid(
    ds_subset,
    coords={
        'X':{'center':'x', 'right':'x_c'},
        'Y':{'center':'y', 'right':'y_c'},
        'Z':{'center':'lev'},
    },
    periodic=False,
    boundary='extend',
    metrics={('X','Y'): 'areacello'}
)

# Step 2: Interpolate the available metric to the desired variable grid and assign it as a coordinate
areacello_uo = grid.interp(ds_subset.areacello,("X"))
ds_subset = ds_subset.assign_coords(areacello_uo=areacello_uo.reset_coords(drop=True).fillna(0))

# Step 3: Create a new grid object
grid_demo = Grid(
    ds_subset,
    coords={
        'X':{'center':'x', 'right':'x_c'},
        'Y':{'center':'y', 'right':'y_c'},
        'Z':{'center':'lev'},
    },
    periodic=False,
    boundary='extend',
    metrics={('X','Y'): 'areacello_uo'}
)

# Step 4: Calculate the average and plot the time series
mean_uo_demo = grid_demo.average(uo,['X','Y'])
mean_uo_demo.plot()
```



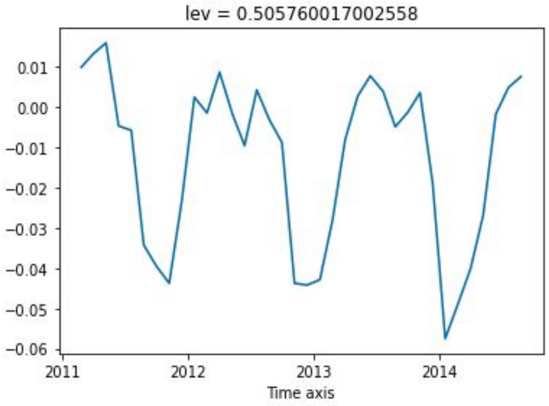
New way = easier, better, faster, and shorter!

```python
import xgcm
xgcm.__version__
```

```
'0.5.2.dev73+g6df944b'
```

```python
mean_uo = grid.average(uo,['X','Y'])
mean_uo.plot()
```

```
/srv/conda/envs/notebook/lib/python3.8/site-packages/xgcm/grid.py:1363: UserWarning: Metric at ('time', 'y', 'x_c') being interpolated from metrics at dimensions ('y', 'x'). Boundary value set to 'extend'.
  warnings.warn(
[<matplotlib.lines.Line2D at 0x7f36a3d26fa0>]
```



grid.average() calls get_metric() to find an existing metric `[areacello]` then automatically interpolates the missing one `[areacello_uo]` using interp_like()

set_metrics lets you assign values to grid objects

```
# Step 1: Assign areacello_uo as a coordinate of subset so that you can assign it as a metric
subset = ds_subset.assign_coords(areacello_uo=areacello_uo.reset_coords(drop=True).fillna(0)) # fill missing values with 0

# Step 2: Create an updated grid object
grid_updated = Grid(
    subset,
    coords={
        'X':{'center':'x', 'right':'x_c'},
        'Y':{'center':'y', 'right':'y_c'},
        'Z':{'center':'lev'},
    },
    periodic=False,
    boundary='extend',
)

# Step 3a: Assign areacello_uo as a metric.
grid_updated.set_metrics(('X','Y'),'areacello_uo')

# Step 3b: Take note that with set_metrics you can assign multiple metrics on the same axes to your dataset as long as they have different dimensions.
grid_updated.set_metrics(('X','Y'),'areacello')

# Step 4: Double check if the metrics were assigned
grid_updated._metrics
```

```
{frozenset({'X',
            'Y'}): [<xarray.DataArray 'areacello_uo' (y: 38, x_c: 20)>
  dask.array<where, shape=(38, 20), dtype=float32, chunksize=(38, 19), chunktype=numpy.ndarray>
  Dimensions without coordinates: y, x_c, <xarray.DataArray 'areacello' (y: 38, x: 20)>
  dask.array<getitem, shape=(38, 20), dtype=float32, chunksize=(38, 20), chunktype=numpy.ndarray>
  Dimensions without coordinates: y, x
  Attributes:
      cell_methods:      area: sum
      description:       Cell areas for any grid used to report ocean variables...
      history:           none
      long_name:         Grid-Cell Area
      online_operation:  once
      standard_name:     cell_area
      units:             m2]}
```

Note: set_metrics gives you flexibility when assigning metrics, but it’s not required to use grid.average()

New updated xgcm = easier, better, faster, and shorter!

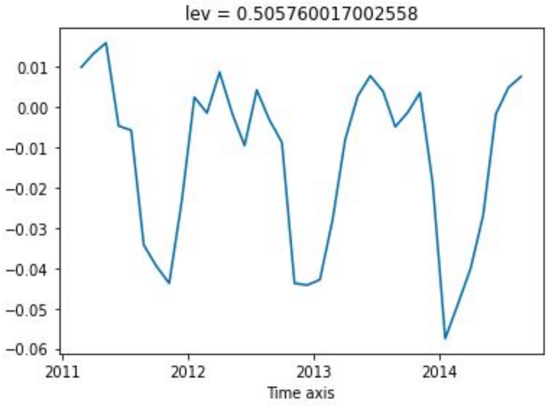
```
import xgcm
xgcm.__version__
```

```
'0.5.2.dev73+g6df944b'
```

```
mean_uo = grid.average(uo,['X','Y'])
mean_uo.plot()
```

```
/srv/conda/envs/notebook/lib/python3.8/site-packages/xgcm/grid.py:1363: UserWarning: Metric at ('time', 'y', 'x_c') being interpolated from metrics at dimensions ('y', 'x'). Boundary value set to 'extend'.
  warnings.warn(
```

```
[<matplotlib.lines.Line2D at 0x7f36a3d26fa0>]
```



Maráming salámat pô!

Interactive Jupyter notebook: bit.ly/xgcm_demo_siparcs2021

grid.average now uses two methods “under the hood”: interp_like and get_metric which can interpolate metrics

interp_like() inputs: available metric, variable you need the metric for

```
areacello_uo = grid.interp_like(ds_subset.areacello,ds_subset.uo)
areacello_uo_getmetric = grid.get_metric(ds_subset.uo,("X","Y"))
```

```
/srv/conda/envs/notebook/lib/python3.8/site-packages/xgcm/grid.py:1363: UserWarning: Metric at ('time', 'lev', 'y', 'x_c') being interpolated from me
trics at dimensions ('y', 'x'). Boundary value set to 'extend'.
  warnings.warn(
```

get_metric() inputs: variable you need the metric for, axes for interpolation