ECCO_v4_A_Better_Method_for_Loading_ECCOv4_NetCDF_Tile_File

February 7, 2018

1 A Better Method for Loading ECCOv4 NetCDF Tile Files

1.1 Objectives:

Introduce an alternative method for loading ECCO v4 NetCDF tile files that returns `Dataset` and `DataArray` objects with better labelling of variable coordinates with respect to *where* they are situated on the Arakawa-C grid.

1.2 Introduction

As we showed in the first tutorial, we can use the `open_dataset` method from `xarray` to load a NetCDF tile file into Python as a `Dataset` object. `open_dataset` is very convienent because it automatically parses the NetCDF file and constructs a `Dataset` object using all of the dimensions, coordinates, variables, and metadata information. However, by default the names of the coordinates are pretty generic: **i1, i2, i3**, etc. We can do a lot better.

In the last tutorial we loaded a single ECCOv4 grid tile file and examined its contents. Let's load it up again and take another look at its coordinates. This time we'll name the new `Dataset` object `grid_3_od` since we are loading the file using `open_dataset`.

```
In [68]: import matplotlib.pylab as plt
         import numpy as np
         import sys
         import xarray as xr
         from copy import deepcopy
         import ecco_v4_py as ecco

In [69]: # point to your local directory holding the nctiles_grid files
         grid_dir='/Volumes/ECCO_BASE/ECCO_v4r3/nctiles_grid/'
         fname = 'GRID.0003.nc'
         grid_3_od = xr.open_dataset(grid_dir + fname)

In [70]: grid_3_od

Out[70]: <xarray.Dataset>
         Dimensions:  (i1: 50, i2: 90, i3: 90)
         Coordinates:
```

```
  * i1       (i1) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
  * i2       (i2) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
  * i3       (i3) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
Data variables:
    hFacC    (i1, i2, i3) float64 ...
    hFacW    (i1, i2, i3) float64 ...
    hFacS    (i1, i2, i3) float64 ...
    XC       (i2, i3) float64 ...
    YC       (i2, i3) float64 ...
    XG       (i2, i3) float64 ...
    YG       (i2, i3) float64 ...
    RAC      (i2, i3) float64 ...
    RAZ      (i2, i3) float64 ...
    DXC      (i2, i3) float64 ...
    DYC      (i2, i3) float64 ...
    DXG      (i2, i3) float64 ...
    DYG      (i2, i3) float64 ...
    Depth    (i2, i3) float64 ...
    AngleCS  (i2, i3) float64 ...
    AngleSN  (i2, i3) float64 ...
    RC       (i1) float64 ...
    RF       (i1) float64 ...
    DRC      (i1) float64 ...
    DRF      (i1) float64 ...
Attributes:
    description:    C-grid parameters (see MITgcm documentation for details)...
    A:              :Format      = native grid (nctiles w. 13 tiles)
    B:              :source      = ECCO consortium (http://ecco-group.org/)
    C:              :institution = JPL/UT/MIT/AER
    D:              :history     = files revision history :
    E:                                 04/20/2017: fill in geometry info for ...
    F:                                 11/06/2016: third release of ECCO v4 (...
    G:                               estimates revision history (from second re...
    H:                                 employs bi-harmonic viscosity (enhance...
    I:                                 sea-ice parameters, updated or novel o...
    J:                                 GRACE OBP, Aquarius SSS, global mean s...
    K:                                 time-series, extended and/or expanded ...
    L:                                 revised weights including data and con...
    M:                                 to account for grid-size variation and...
    N:                                 separate time-mean and time-variable d...
    O:                                 and controls, sea-ice costs, and initi...
    P:                                 additional controls.\n
    Q:              :references  = Forget, G., J.-M. Campin, P. Heimbach, C. ...
    R:                              and C. Wunsch, 2015: ECCO version 4: an i...
    S:                              non-linear inverse modeling and global oc...
    T:                              Geoscientific Model Development, 8, 3071-...
    U:                             Forget, G., J.-M. Campin, P. Heimbach, C. ...
    V:                              ECCO version 4: Second Release, 2016, htt...
```

```
W:              file created using gcmfaces_IO/write2nctiles.m
date:           21-Apr-2017
Conventions:    CF-1.6
_FillValue:     nan
missing_value:  nan
```

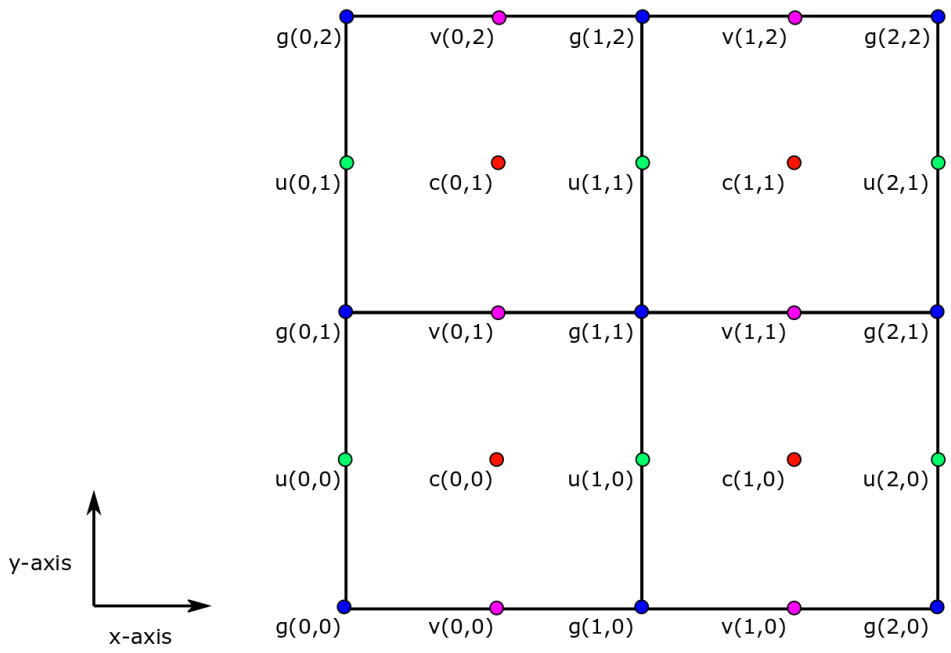
1.2.1 Examining the Dataset object contents

We see that all of the Data variables in `grid_3_od` use one of three dimensions, **i1**, **i2**, and **i3**. As we saw before, some variables are 3D, $hFacC(i1, i2, i3)$; others are 2D, $XC(i2, i3)$; and others are 1D, $RF(i1)$.

While this `Dataset` object is useful, its coordinate names are somewhat ambiguous. The MITgcm uses the staggered Arakawa-C grid (hereafter c-grid) to discretize the model equations. Model variables are not co-located in c-grid models; they can fall on one of four different categories of point. From the above, one cannot distinguish *where* on the *c-grid* these different variables are situated.

To understand this issue better, let's review the c-grid coordinate system.

1.3 The four horizontal points of the c-grid



The four different categories of point used in the staggered Arakawa-C grid (C-grid)

1.3.1 *c* points

Scalar variables (e.g., $T, S, SSH, OBP, SIarea$) and variables associated with vertical velocity (e.g., $WVEL$) are situated at the center of the tracer grid cell in the horizontal plane. These are c points

Define the coordinates (i, j) for the discrete indices of c points in the x and y directions, respectively.

In the ECCO v4 NetCDF files, $c(0, 0)$ is the $-x$ most and $-y$ most tracer grid cell.

- In the $+y$ direction, the next c point is $c(0, 1)$.
- In the $+x$ direction, the next c point is $c(1, 0)$

1.3.2 u points

Vector variables related to horizontal velocity in the x direction are staggered along the edges of tracer cells between c points in the horizontal plane. Examples include horizontal velocity in the x direction ($UVEL$) and horizontal advective flux of snow in the x direction ($ADVxSNOW$). They are situated along the edges (if 2D) or faces (if 3D) of the tracer grid cells in the x direction.

Define the coordinates (i_g, j) for the discrete indices of u points in the x and y directions, respectively.

We use i_g as the coordinate in the x direction because u points are situated along the tracer grid cell ed***G***es. We use j for its y coordinate because u points and c points fall along the same lines in y.

In the ECCO v4 netCDF files, $u(0, 0)$ is the $-x$ most and $-y$ most u point.

1.3.3 v points

Vector variables related to horizontal velocity in the y direction are staggered along the edges of tracer cells between c points in the horizontal plane. Examples include horizontal velocity in the y direction ($VVEL$) and horizontal advective flux of snow in the y direction ($ADVySNOW$). They are situated along the edges (if 2D) or faces (if 3D) of the tracer grid cells in the y direction.

Define the coordinates (i, j_g) for the discrete indices of v points in the x and y directions, respectively.

We use j_g as the coordinate in the y direction because v points are situated along the tracer grid cell ed***G***es. We use i for its x coordinate because v points and c points fall along the same lines in x.

In the ECCO v4 NetCDF files, $v(0, 0)$ is the $-x$ most and $-y$ most v point.

1.3.4 g points

Variables that are explictly related to horizontal velocities in the model in both the x and y direction are situated at g points in the horizontal plane. g points are situated at the corners of tracer grid cells.

Define the coordinates (i_g, j_g) for the discrete indices of g points in the x and y directions, respectively.

We use i_g and j_g because g points are on the ed***G***es (corners) of tracer grid cells.

In the ECCO v4 NetCDF files, $g(0, 0)$ is the $-x$ most and $-y$ most g point.

1.4 The two vertical points of the c-grid

There are two different coordinates in the vertical z dimension:

1.4.1 w points

Variables related to vertical velocity or vertical fluxes are situated at w points in the vertical direction. These variables are situated on the upper and lower faces of the tracer grid cell.

Define the coordinate k_g for w points using the same reasoning as above: w points fall along the the ed***G***es of tracer grid cells in the z direction (top and bottom of the grid cells).

In the ECCO v4 NetCDF files, $k_g(0)$ is the sea surface.

1.4.2 *k* points

Variables that are not related to vertical velocity or vertical fluxes are situated at k points in the vertical direction. These variables are situated on the upper and lower faces of the tracer grid cell.

Define the coordinate k for points situated in the center of the grid cells.

In the ECCO v4 NetCDF files, $k(0)$ is the middle of the uppermost tracer grid cell.

1.5 Applying the C-grid coordinates to the variables

The default coordinate names in the ECCO v4 netcdf tile files do not distinguish distinguish between the four horizontal coordinates, i, i_g, j, j_g and the two vertical coordinates, k_g and k, used by our c-grid model.

To apply these more descriptive coordinates to the `Dataset` objects that are created when we load netCDF files, we provide a special routine, `load_tile_from_netcdf`.

1.5.1 `load_tile_from_netcdf`

This routine takes four arguments, 1. *data_dir*: the directory of the netCDF file 2. *var*: the name of the variable stored in the netCDF file (without the tile number). Examples include *THETA*, *VVEL*, *DFxEHFF* 3. *var_type*: one of 'c','g','u','v', or 'grid' corresponding with the variable's c-grid point type. 'grid' is a special case because unlike every other variable NetCDF file, grid files actually contain several variables that are on different c-grid points. 4. *tile_index*: the tile number [1 .. 13]

1.5.2 Loading an ECCO v4 netCDF grid tile file using `load_tile_from_netcdf`

Let's use `load_tile_from_netcdf` to load grid tile 3 again. This time we'll call the new `Dataset` object `grid_3_new`

```
In [71]: var = 'GRID'
         var_type = 'grid'
         tile_index = 3
         grid_3_new = ecco.load_tile_from_netcdf(grid_dir, var, var_type, tile_index)

loading /Volumes/ECCO_BASE/ECCO_v4r3/nctiles_grid/GRID.0003.nc


In [72]: grid_3_new

Out[72]: <xarray.Dataset>
         Dimensions:  (i: 90, i_g: 90, j: 90, j_g: 90, k: 50, k_g: 50)
         Coordinates:
             tile     int64 3
           * k        (k) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
           * i        (i) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
           * j        (j) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
           * i_g      (i_g) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 ...
           * j_g      (j_g) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 ...
```

```
  * k_g      (k_g) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 ...
Data variables:
    XC       (j, i) float64 ...
    YC       (j, i) float64 ...
    RAC      (j, i) float64 ...
    Depth    (j, i) float64 ...
    AngleCS  (j, i) float64 ...
    AngleSN  (j, i) float64 ...
    hFacC    (k, j, i) float64 ...
    land_c   (k, j, i) float64 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 ...
    XG       (j_g, i_g) float64 ...
    YG       (j_g, i_g) float64 ...
    RAZ      (j_g, i_g) float64 ...
    DXC      (j, i_g) float64 ...
    DYG      (j, i_g) float64 ...
    hFacW    (k, j, i_g) float64 ...
    land_u   (k, j, i_g) float64 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 ...
    DYC      (j_g, i) float64 ...
    DXG      (j_g, i) float64 ...
    hFacS    (k, j_g, i) float64 ...
    land_v   (k, j_g, i) float64 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 ...
    RF       (k_g) float64 ...
    DRC      (k_g) float64 ...
    RC       (k) float64 ...
    DRF      (k) float64 ...
Attributes:
    description:    C-grid parameters (see MITgcm documentation for details)...
    A:              :Format      = native grid (nctiles w. 13 tiles)
    B:              :source      = ECCO consortium (http://ecco-group.org/)
    C:              :institution = JPL/UT/MIT/AER
    D:              :history     = files revision history :
    E:                                 04/20/2017: fill in geometry info for ...
    F:                                 11/06/2016: third release of ECCO v4 (...
    G:                               estimates revision history (from second re...
    H:                                 employs bi-harmonic viscosity (enhance...
    I:                                 sea-ice parameters, updated or novel o...
    J:                                 GRACE OBP, Aquarius SSS, global mean s...
    K:                                 time-series, extended and/or expanded ...
    L:                                 revised weights including data and con...
    M:                                 to account for grid-size variation and...
    N:                                 separate time-mean and time-variable d...
    O:                                 and controls, sea-ice costs, and initi...
    P:                                 additional controls.\n
    Q:              :references  = Forget, G., J.-M. Campin, P. Heimbach, C. ...
    R:                              and C. Wunsch, 2015: ECCO version 4: an i...
    S:                              non-linear inverse modeling and global oc...
    T:                              Geoscientific Model Development, 8, 3071-...
    U:                             Forget, G., J.-M. Campin, P. Heimbach, C. ...
```

```
V:                              ECCO version 4: Second Release, 2016, htt...
W:              file created using gcmfaces_IO/write2nctiles.m
date:           21-Apr-2017
Conventions:    CF-1.6
_FillValue:     nan
missing_value:  nan
```

1.5.3 Examining the Dataset object contents

1. Dimensions `Dimensions: (i: 90, i_g: 90, j: 90, j_g: 90, k: 50, k_g: 50)`

The *Dimensions* list now lists the six different coordinates and their dimension used by variables stored in this new grid tile `Dataset` object.

2. Coordinates

```
Coordinates:
    tile     int64 3
  * k        (k) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
  * i        (i) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
  * j        (j) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
  * i_g      (i_g) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 ...
  * j_g      (j_g) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 ...
  * k_g      (k_g) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 ...
```

Each of the 6 dimensions has a corresponding set of coordinate labels.

3. Data Variables

```
Data variables:
    XC       (j, i) float64 ...
    hFacC    (k, j, i) float64 ...
    XG       (j_g, i_g) float64 ...
    DXC      (j, i_g) float64 ...
    hFacW    (k, j, i_g) float64 ...
    DYC      (j_g, i) float64 ...
    hFacS    (k, j_g, i) float64 ...
    RF       (k_g) float64 ...
    RC       (k) float64 ...
```

We now see that each *data variable* is now described with its proper coordinates. For example, XC, the longitude of the tracer grid cell center uses i, j coordiantes while XG, the longitude grid cell corners, now has i_g, j_g coordinates. DXC, the distance between adjacent cell centers, uses i_g, j coordiantes: consistent with the notion that distances between cell centers in x are situated halfway between cell centers.

Note: The ordering of arrays is not (x,y,z) but (z,y,x).

In addition to the variables now having more descriptive coordinates, the `load_tile_from_netcdf` routine also adds 3 new three-dimensional land masks, one each for grid cell at the 'u' (land_u), 'c' (land_c), and 'v' (land_v) points.

1.6 A closer look at the DataArray after loading with `load_tile_from_netcdf`

Let's take a quick look at one of the variables loaded using the `load_tile_from_netcdf` routine:

```
In [73]: grid_3_new.XC
```

```
Out[73]: <xarray.DataArray 'XC' (j: 90, i: 90)>
         array([[-37.5     , -36.5     , -35.5     , ...,  49.5     ,  50.5     ,  51.5     ],
                [-37.5     , -36.5     , -35.5     , ...,  49.5     ,  50.5     ,  51.5     ],
                [-37.5     , -36.5     , -35.5     , ...,  49.5     ,  50.5     ,  51.5     ],
                ...,
                [-37.730072, -37.178291, -36.597565, ...,  50.597565,  51.178291,
                  51.730072],
                [-37.771988, -37.291943, -36.764027, ...,  50.764027,  51.291943,
                  51.771988],
                [-37.837925, -37.44421 , -36.968143, ...,  50.968143,  51.44421 ,
                  51.837925]])
         Coordinates:
             tile     int64 3
           * i        (i) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
           * j        (j) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
         Attributes:
             long_name:          longitude
             units:              degrees_east
             rotated_to_latlon:  False
```

The `load_tile_from_netcdf` routine adds a new coordinate to the variables, *tile*. This will be helpful when loading and combining multiple tiles.

Finally, we see a new attribute: *rotated_to_latlon*. This attribute is a flag telling us that the arrays are in the original lat-lon-cap tile layout. We'll discuss this later.

1.7 Loading non-grid ECCO variables using `load_tile_from_netcdf`

So far we've only looked at ECCO v4 grid tile files. With the `load_tile_from_netcdf` routine you can assign the proper coordinates to any variable by specifying its c-grid point category.

We'll demonstrate by loading three variables, one each that are on 'c','u', and 'v' points

1.7.1 A 'c' point variable: SSH

Let's load tile 3 of the 'c' point sea surface height (SSH) variable.

```
In [74]: data_dir='/Volumes/ECCO_BASE/ECCO_v4r3/nctiles_monthly/SSH/'
         var = 'SSH'
         var_type = 'c'
         tile_index = 3
         ssh_tile_3 = ecco.load_tile_from_netcdf(data_dir, var, var_type, tile_index)
```

```
loading /Volumes/ECCO_BASE/ECCO_v4r3/nctiles_monthly/SSH/SSH.0003.nc
```

Since this is the first time we're loading an output variable from the state estimate let's closely examine `ssh_tile_3`.

```
In [75]: ssh_tile_3
```

```
Out[75]: <xarray.Dataset>
         Dimensions:    (i: 90, j: 90, tile: 1, time: 288)
         Coordinates:
           * time       (time) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 ...
           * j          (j) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
           * i          (i) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
             lon_c      (j, i) float64 -37.5 -36.5 -35.5 -34.5 -33.5 -32.5 -31.5 -30.5 ...
             lat_c      (j, i) float64 10.46 10.46 10.46 10.46 10.46 10.46 10.46 10.46 ...
             tim        (time) datetime64[ns] ...
           * tile       (tile) int64 3
             timestep   (time) float64 732.0 1.428e+03 2.172e+03 2.892e+03 3.636e+03 ...
         Data variables:
             SSH        (time, tile, j, i) float64 0.0576 0.05474 0.05277 0.05157 ...
         Attributes:
             description:    SSH -- ECCO v4 ocean state estimate, release 3 -- 1992-2015
             A:              :Format      = native grid (nctiles w. 13 tiles)
             B:              :source      = ECCO consortium (http://ecco-group.org/)
             C:              :institution = JPL/UT/MIT/AER
             D:              :history     = files revision history :
             E:                                  05/02/2017: adjust SSH and OBP with gl...
             F:                                  05/02/2017: adjust SSH with sea-ice lo...
             G:                                  04/20/2017: fill in geometry info for ...
             H:                                  11/06/2016: third release of ECCO v4 (...
             I:                                estimates revision history (from second re...
             J:                                  employs bi-harmonic viscosity (enhance...
             K:                                  sea-ice parameters, updated or novel o...
             L:                                  GRACE OBP, Aquarius SSS, global mean s...
             M:                                  time-series, extended and/or expanded ...
             N:                                  revised weights including data and con...
             O:                                  to account for grid-size variation and...
             P:                                  separate time-mean and time-variable d...
             Q:                                  and controls, sea-ice costs, and initi...
             R:                                  additional controls.\n
             S:              :references  = Forget, G., J.-M. Campin, P. Heimbach, C. ...
             T:                              and C. Wunsch, 2015: ECCO version 4: an i...
             U:                              non-linear inverse modeling and global oc...
             V:                              Geoscientific Model Development, 8, 3071-...
             W:                            Forget, G., J.-M. Campin, P. Heimbach, C. ...
             X:                              ECCO version 4: Second Release, 2016, htt...
             Y:              file created using gcmfaces_IO/write2nctiles.m
             date:           02-May-2017
             Conventions:    CF-1.6
             _FillValue:     nan
             missing_value:  nan
```

Notice that unlike the grid files that we've loaded up until now, the `ssh_tile_3` `Dataset` object only has one *Data variable*, *SSH*. Non-grid ECCO v4 NetCDF tile files only ever store *one* physical variable. Consequently, the *dimensions* of the `Dataset` will be the same as the *dimensions* of the single *data variable*. Let's take a look at the *SSH* `DataArary`:

```
In [76]: ssh_tile_3.SSH
```

```
Out[76]: <xarray.DataArray 'SSH' (time: 288, tile: 1, j: 90, i: 90)>
         array([[[[ 0.0576  , ...,  0.383994],
                  ...,
                  [      nan, ...,       nan]]],


                ...,
                [[[ 0.075663, ...,  0.458817],
                  ...,
                  [      nan, ...,       nan]]]])
         Coordinates:
           * time      (time) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 ...
           * j         (j) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
           * i         (i) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
             lon_c     (j, i) float64 -37.5 -36.5 -35.5 -34.5 -33.5 -32.5 -31.5 -30.5 ...
             lat_c     (j, i) float64 10.46 10.46 10.46 10.46 10.46 10.46 10.46 10.46 ...
             tim       (time) datetime64[ns] ...
           * tile      (tile) int64 3
             timestep  (time) float64 732.0 1.428e+03 2.172e+03 2.892e+03 3.636e+03 ...
         Attributes:
             long_name:          Surface Height Anomaly adjusted with global steric he...
             units:              m
             rotated_to_latlon:  False
```

Dimensional coordinates As expected, the `SSH` `DataArray` uses **i, j** coordinates for its horizontal dimensions. We also see a **tile** and **time** coordinates. The ordering of the *SSH* dimensions is **time, tile, j, i** with the logic being that **tile** is a kind of space coordinate.

We find 288 records in the **time** dimensions. One for each month of the 1992-2015 state estimate. There is one record in the **tile** dimensions because we have only loaded a single tile file so far.

Non-dimensional coordinates Notice the four *Coordinates* that do not have an "*" in front of their names. These are so-called non-dimensional coordinates. Think of non-dimensional coordinates as helpful extra coordinate labels. Here, the non-dimensional coordinates include two for the **time** dimension: **tim** and **timestep**, and two for the space dimensions: **lon_c, lat_c**.

The time non-dimensional coordinates

```
In [77]: ssh_tile_3.time
```

```
Out[77]: <xarray.DataArray 'time' (time: 288)>
         array([   1.,    2.,    3., ...,  286.,  287.,  288.])
         Coordinates:
           * time      (time) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 ...
             tim       (time) datetime64[ns] ...
             timestep  (time) float64 732.0 1.428e+03 2.172e+03 2.892e+03 3.636e+03 ...
         Attributes:
             long_name:  array index 1
             units:      1
```

The **tim** non-dimensional coordinate provides the calendar date (centered in the month) for the **time** monthly index and the **timestep** non-dimensional coordinate provides the the timestep number corresponding to each month (the ECCO v4 simulation uses 1-hourly timesteps).

The space non-dimensional coordinates **lon_c** and **lat_c** non-dimensional coordinates provide the longitude and latitude for each of the 'c' points of SSH.

```
In [78]: ssh_tile_3.lon_c
```

```
Out[78]: <xarray.DataArray 'lon_c' (j: 90, i: 90)>
         array([[-37.5     , -36.5     , -35.5     , ...,  49.5     ,  50.5     ,  51.5     ],
                [-37.5     , -36.5     , -35.5     , ...,  49.5     ,  50.5     ,  51.5     ],
                [-37.5     , -36.5     , -35.5     , ...,  49.5     ,  50.5     ,  51.5     ],
                ...,
                [-37.730072, -37.178291, -36.597565, ...,  50.597565,  51.178291,
                  51.730072],
                [-37.771988, -37.291943, -36.764027, ...,  50.764027,  51.291943,
                  51.771988],
                [-37.837925, -37.44421 , -36.968143, ...,  50.968143,  51.44421 ,
                  51.837925]])
         Coordinates:
           * j         (j) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
           * i         (i) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
             lon_c     (j, i) float64 -37.5 -36.5 -35.5 -34.5 -33.5 -32.5 -31.5 -30.5 ...
             lat_c     (j, i) float64 10.46 10.46 10.46 10.46 10.46 10.46 10.46 10.46 ...
         Attributes:
             units:          degrees_east
             standard_name:  longitude
```

1.7.2 A 'u' point variable: UVEL

Let's load tile 3 of the 'u' point horizontal velocity in the local x direction variable, $UVEL$.

```
In [79]: data_dir='/Volumes/ECCO_BASE/ECCO_v4r3/nctiles_monthly/UVEL/'
         var = 'UVEL'
         var_type = 'u'
         tile_index = 3
         uvel_tile_3 = ecco.load_tile_from_netcdf(data_dir, var, var_type, tile_index)
```

```
loading /Volumes/ECCO_BASE/ECCO_v4r3/nctiles_monthly/UVEL/UVEL.0003.nc
```

Let's look at `uvel_tile_3`. This time let's also remove some of the descriptive NetCDF file *Attributes* using a little routine called `minimal_metadata`. We've already seen these attributes a number of times.

```
In [80]: ecco.minimal_metadata(uvel_tile_3)

Removing Dataset Attributes A-Z
```

```
In [81]: uvel_tile_3

Out[81]: <xarray.Dataset>
         Dimensions:   (i_g: 90, j: 90, k: 50, tile: 1, time: 288)
         Coordinates:
           * time      (time) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 ...
           * k         (k) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
           * j         (j) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
           * i_g       (i_g) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 ...
             lon_u     (j, i_g) float64 -38.0 -37.0 -36.0 -35.0 -34.0 -33.0 -32.0 ...
             lat_u     (j, i_g) float64 10.46 10.46 10.46 10.46 10.46 10.46 10.46 ...
             dep       (k) float64 ...
             tim       (time) datetime64[ns] ...
           * tile      (tile) int64 3
             timestep  (time) float64 732.0 1.428e+03 2.172e+03 2.892e+03 3.636e+03 ...
         Data variables:
             UVEL      (time, tile, k, j, i_g) float64 -0.02911 -0.02432 -0.02008 ...
         Attributes:
             description:    UVEL -- ECCO v4 ocean state estimate, release 3 -- 1992-2015
             date:           03-Nov-2017
             Conventions:    CF-1.6
             _FillValue:     nan
             missing_value:  nan
```

`uvel_tile_3` has one *Data variable, UVEL*. Let's take a look at the *UVEL* `DataArary`:

```
In [82]: uvel_tile_3.UVEL

Out[82]: <xarray.DataArray 'UVEL' (time: 288, tile: 1, k: 50, j: 90, i_g: 90)>
         array([[[[[-0.029109, ...,       nan],
                   ...,
                   [      nan, ...,       nan]],

                  ...,
                  [[      nan, ...,       nan],
                   ...,
```

```
           [      nan, ...,       nan]]]],


       ...,
       [[[[-0.050949, ...,       nan],
          ...,
          [      nan, ...,       nan]],

         ...,
         [[      nan, ...,       nan],
          ...,
          [      nan, ...,       nan]]]])
Coordinates:
  * time      (time) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 ...
  * k         (k) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
  * j         (j) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
  * i_g       (i_g) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 ...
    lon_u     (j, i_g) float64 -38.0 -37.0 -36.0 -35.0 -34.0 -33.0 -32.0 ...
    lat_u     (j, i_g) float64 10.46 10.46 10.46 10.46 10.46 10.46 10.46 ...
    dep       (k) float64 ...
    tim       (time) datetime64[ns] ...
  * tile      (tile) int64 3
    timestep  (time) float64 732.0 1.428e+03 2.172e+03 2.892e+03 3.636e+03 ...
Attributes:
    long_name:          Zonal Component of Velocity (m/s)
    units:              m/s
    rotated_to_latlon:  False
```

Dimensional coordinates As expected, *UVEL* uses the **i_g, j** coordinates for its horizontal dimensions. Unlike *SSH*, *UVEL* has three-dimensions in space so we find a **k** coordinate. The ordering of the three-dimensional ECCO v4 output is **time, tile, k, j, i**.

Non-dimensional coordinates *UVEL* has one new non-dimensional coordinate for **k**: **dep** and two new non-dimensional coordinates for space: **lon_u, lat_u**

The dep non-dimensional coordinates **dep** is the depth of the center of the tracer grid cell in meters:

```
In [83]: uvel_tile_3.dep
```

```
Out[83]: <xarray.DataArray 'dep' (k: 50)>
         array([  5.000000e+00,   1.500000e+01,   2.500000e+01,   3.500000e+01,
                  4.500000e+01,   5.500000e+01,   6.500000e+01,   7.500500e+01,
                  8.502500e+01,   9.509500e+01,   1.053100e+02,   1.158700e+02,
                  1.271500e+02,   1.397400e+02,   1.544700e+02,   1.724000e+02,
                  1.947350e+02,   2.227100e+02,   2.574700e+02,   2.999300e+02,
```

```
                3.506800e+02,   4.099300e+02,   4.774700e+02,   5.527100e+02,
                6.347350e+02,   7.224000e+02,   8.144700e+02,   9.097400e+02,
                1.007155e+03,   1.105905e+03,   1.205535e+03,   1.306205e+03,
                1.409150e+03,   1.517095e+03,   1.634175e+03,   1.765135e+03,
                1.914150e+03,   2.084035e+03,   2.276225e+03,   2.491250e+03,
                2.729250e+03,   2.990250e+03,   3.274250e+03,   3.581250e+03,
                3.911250e+03,   4.264250e+03,   4.640250e+03,   5.039250e+03,
                5.461250e+03,   5.906250e+03])
        Coordinates:
          * k        (k) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
            dep      (k) float64 5.0 15.0 25.0 35.0 45.0 55.0 65.0 75.0 85.03 95.1 ...
        Attributes:
            units:          m
            standard_name:  depth
```

The space non-dimensional coordinates **lon_u** and **lat_u** provide the longitude and latitude for each **i_g,j** point.

1.7.3 A 'v' point variable: VVEL

Let's load tile 3 of the 'v' point horizontal velocity in the local x direction variable, $VVEL$.

```
In [84]: data_dir='/Volumes/ECCO_BASE/ECCO_v4r3/nctiles_monthly/VVEL/'
         var = 'VVEL'
         var_type = 'v'
         tile_index = 3
         vvel_tile_3 = ecco.load_tile_from_netcdf(data_dir, var, var_type, tile_index)
         ecco.minimal_metadata(vvel_tile_3)
```

```
loading /Volumes/ECCO_BASE/ECCO_v4r3/nctiles_monthly/VVEL/VVEL.0003.nc
Removing Dataset Attributes A-Z
```

Let's look at the *Coordinates* of $VVEL$.

```
In [85]: vvel_tile_3.coords
```

```
Out[85]: Coordinates:
           * time      (time) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 ...
           * k         (k) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
           * j_g       (j_g) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 ...
           * i         (i) float64 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 ...
             lon_v     (j_g, i) float64 -37.5 -36.5 -35.5 -34.5 -33.5 -32.5 -31.5 ...
             lat_v     (j_g, i) float64 9.97 9.97 9.97 9.97 9.97 9.97 9.97 9.97 9.97 ...
             dep       (k) float64 ...
             tim       (time) datetime64[ns] ...
           * tile      (tile) int64 3
             timestep  (time) float64 732.0 1.428e+03 2.172e+03 2.892e+03 3.636e+03 ...
```

Dimensional coordinates As expected, `vvel_tile_3` uses the **i, j_g** coordinates for its horizontal dimensions.

Non-dimensional coordinates $VVEL$ has two new non-dimensional coordinates for space: **lon_v, lat_v**. As you might expect, these are the longitude and latitude of this v-point variable.

1.8 Conclusion

Now you know how to load variables so that their coordinates correspond with their location on the c-grid.