ECCO Resources and Tools

Overview ECCO V4 datasets, Python & Julia tools and EMU

- Ian Fenty (JPL)
- Ichiro Fukumori (JPL)
- Gael Forget (MIT)

ECCO V4 Py: Python Package

An open-source resource to facilitate analysis of ECCO V4

Subroutines for:

- Global mean sea level, ocean heat content, ocean volume
- Global and regional budgets for volume, heat, salt, salinity, and freshwater
- Volume, heat, and salt transports across arbitrary sections
- Meridional overturning circulation streamfunction
- Plotting llc-grid fields

Leverages other open-source projects

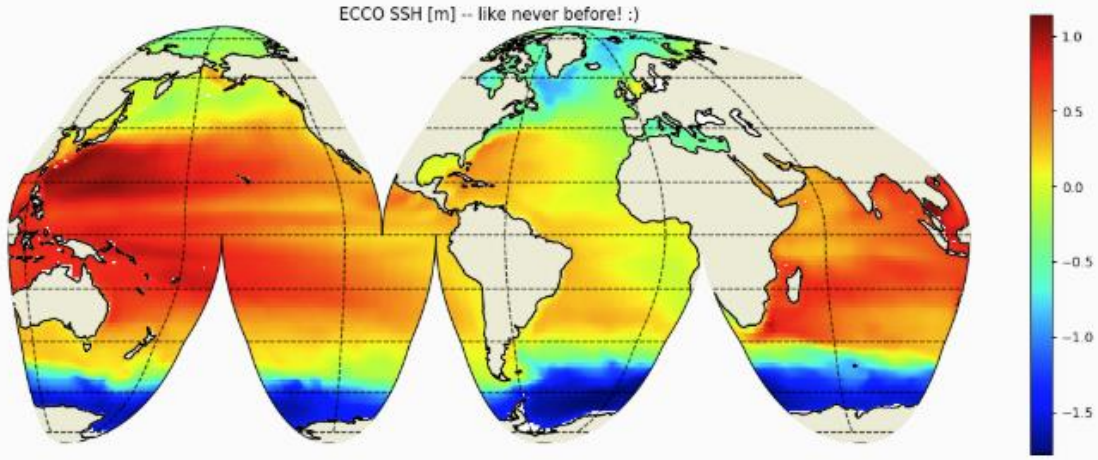
- xgcm, xmitgcm, xarray, cartopy, ect.

[27]:
```
plt.figure(figsize=(16,6), dpi=90)

tmp_plt = ecco_ds.SSH.isel(time=1)
tmp_plt = tmp_plt.where(ecco_ds.hFacC.isel(k=0) !=0)

ecco.plot_proj_to_latlon_grid(ecco_ds.XC, ecco_ds.YC, \
                              tmp_plt, \
                              user_lon_0=-66,\
                              projection_type='InterruptedGoodeHomolosine',\
                              plot_type = 'pcolormesh', \
                              show_colorbar=True,
                              dx=1, dy=1);

plt.title('ECCO SSH [m] -- like never before! :)');
```



https://github.com/ECCO-GROUP/ECCOv4-py

[5]: ds

[5]: xarray.Dataset

▸ Dimensions: (**i**: 90, **i_g**: 90, **j**: 90, **j_g**: 90, **k**: 50, **k_u**: 50, **k_l**: 50, **k_p1**: 51, **tile**: 13, **time**: 12, nv: 2, nb: 4)

▾ Coordinates:

i	(i)	int32	0 1 2 3 4 5 6 ... 84 85 86 87 ...
i_g	(i_g)	int32	0 1 2 3 4 5 6 ... 84 85 86 87 ...
j	(j)	int32	0 1 2 3 4 5 6 ... 84 85 86 87 ...
j_g	(j_g)	int32	0 1 2 3 4 5 6 ... 84 85 86 87 ...
k	(k)	int32	0 1 2 3 4 5 6 ... 44 45 46 47 ...
k_u	(k_u)	int32	0 1 2 3 4 5 6 ... 44 45 46 47 ...
k_l	(k_l)	int32	0 1 2 3 4 5 6 ... 44 45 46 47 ...
k_p1	(k_p1)	int32	0 1 2 3 4 5 6 ... 45 46 47 48 ...
tile	(tile)	int32	0 1 2 3 4 5 6 7 8 9 10 11 12
time	(time)	datetime64[ns]	2010-01-16T12:00:00 ... 20...
XC	(tile, j, i)	float32	dask.array<chunksize=(13, 9...
YC	(tile, j, i)	float32	dask.array<chunksize=(13, 9...
XG	(tile, j_g, i_g)	float32	dask.array<chunksize=(13, 9...
YG	(tile, j_g, i_g)	float32	dask.array<chunksize=(13, 9...
Z	(k)	float32	dask.array<chunksize=(50,), ...
Zp1	(k_p1)	float32	dask.array<chunksize=(51,), ...
Zu	(k_u)	float32	dask.array<chunksize=(50,), ...
Zl	(k_l)	float32	dask.array<chunksize=(50,), ...
time_bnds	(time, nv)	datetime64[ns]	dask.array<chunksize=(1, 2), ...
XC_bnds	(tile, j, i, nb)	float32	dask.array<chunksize=(13, 9...
YC_bnds	(tile, j, i, nb)	float32	dask.array<chunksize=(13, 9...
Z_bnds	(k, nv)	float32	dask.array<chunksize=(50, 2)...

▾ Data variables:

THETA	(time, k, tile, j, i)	float32	dask.array<chunksize=(1, 25,...
SALT	(time, k, tile, j, i)	float32	dask.array<chunksize=(1, 25,...

▸ Indexes: (10)

▸ Attributes: (62)

Compatible with “raw model output” or prepared netcdf files

Strongly leverages **xarray** labeled data structure.

- Broadcasting: mathematical operations across fields of the same dimension
- Calculations of time-mean, climatologies, anomalies
- Subsampling, masking

ECCO v4 Python Tutorial

http://ecco-v4-python-tutorial.readthedocs.io/

Welcome to the ECCO Version 4 Tutorial

This website contains a set of tutorials about how to use the ECCO Central Production Version 4 (ECCO v4) global ocean and sea-ice state estimate. The tutorials were written in Python and make use of the ecco_v4_py Python library, a library written specifically for loading, plotting, and analyzing ECCO v4 state estimate fields.

Additional Resources

The ECCO v4 state estimate is the output of a free-running simulation of a global ca. 1-degree configuration of the MITgcm. Prior to public release, the model output files model are assembled into NetCDF files. If you would like to work directly with the flat binary "MDS" files provided by the model then take a look at the xmitgcm Python package. The xgcm Python package provides tools for operating on model output fields loaded with xmitgcm. If you wish to analyze the MITgcm model output using Matlab then we strongly recommend the extensive set of tools provided by the gcmfaces toolbox.

The ecco_v4_py package used in this tutorial was inspired by both the xmitgcm package and gcmfaces toolbox.

Getting Started

- The ECCO Ocean and Sea-Ice State Estimate
- ECCO v4 state estimate ocean, sea-ice, and atmosphere fields
- Python and Python Packages
- How to get the ECCO v4 State Estimate
- Tutorial Overview

ECCO Version 4 Tutorial

Getting started
The ECCO Ocean and Sea-Ice State Estimate
ECCO v4 state estimate ocean, sea-ice, and atmosphere fields
Python and Python Packages
Using Python to Download ECCO Datasets
Downloading Subsets of ECCO Datasets
Using *wget* to Download ECCO Datasets from PO.DAAC
AWS Cloud: getting started and retrieving ECCO datasets
Tutorial Overview

ECCO data structures
The Dataset and DataArray objects used in the ECCOv4 Python package
Coordinates and Dimensions of ECCOv4 NetCDF files

Input/output, data structure manipulation
Loading the ECCOv4 native model grid parameters
Loading the ECCOv4 state estimate fields on the native model grid
ECCOv4 Loading llc binary files in the 'compact' format
Combining multiple Datasets
Saving Datasets and DataArrays to NetCDF

Operating on data variables
Accessing and Subsetting Variables
Operating on Numpy arrays

Plotting & interpolation
Plotting Tiles
Interpolating fields from the model llc grid to a regular lat lon grid

Scalar and vector calculations
Example calculations with scalar quantities
Calculating gradients and curl on the ECCO native grid

Intro to PO Tutorials
Intro to PO Tutorials: Getting Started
Part 1: Geostrophic balance
Part 2: Thermal Wind
Part 3: Steric height

More advanced calculations
Compute meridional heat transport
Compute MOC along the approximate OSNAP array from ECCO
ECCOv4 Global Volume Budget Closure
Global Heat Budget Closure
Salt, Salinity and Freshwater Budgets
Calculate ocean thermal forcing from ECCOv4r4 data, direct from PO.DAAC S3 storage

ECCO v4 computations

The ECCO version 4 Python Tutorial website has a wide range of tutorials to help users get started with accessing and using ECCOv4 output. These tutorials cover subjects ranging from loading data files using Python's *xarray* package, to more advanced computations like the steric height, meridional heat transport, and budgets.

Tutorials hosted in `ecco-2025`

A subset of these tutorials have been copied over to the `ecco-2025` Jupyter book and can be accessed below, to give you a sense of their range and structure. The tutorials hosted here already have settings enabled for cloud access, and will use the `efs_ecco` mounted drive for any downloads.

Tutorial	Topics
The Dataset and DataArray objects used in the ECCOv4 Python package	*xarray* intro with ECCOv4 output
Coordinates and Dimensions of ECCOv4 NetCDF files	Arakawa C-grid and ECCOv4 coordinates
Accessing and Subsetting Variables	*xarray* accessing and subsetting
Memory management in Python	Memory management when using *numpy*, *dask*, and *xarray*
Example calculations with scalar quantities	Sample calculations (e.g., SSH global mean)
Calculating gradients and curl on the ECCO native grid	Horizontal derivatives on the C-grid
Steric height	Steric height and thermosteric/halosteric components
Compute meridional heat transport	Meridional heat transport
ECCOv4 Global Volume Budget Closure	Volume budget (minus global mean steric, IB, sea-ice loading)
Energy imbalance in the ocean	The role of the ocean in Earth's energy imbalance

Budget closures of Potential Temperature, Salt, Salinity, Freshwater, and Volume,

ECCOv4 Global Volume Budget Closure

Here we demonstrate the closure of volume budgets in ECCOv4 configurations. This notebook is draws heavily from `evaluating_budgets_in_eccov4r3.pdf` which explains the procedure with Matlab code examples by Christopher G. Piecuch. See ECCO Version 4 release documents: */doc/evaluating_budgets_in_eccov4r3.pdf*).

Objectives

Illustrate how volume budgets are closed globally.

Introduction

ECCOv4 uses the z^* coordinate system in which the depth of the vertical coordinate, z^* varies with time as:

$$z^* = \frac{z - \eta(x,y,t)}{H(x,y) + \eta(x,y,t)} H(x,y)$$

With H being the model depth, η being the model sea level anomaly, and z being depth.

If the vertical coordinate didn't change through time then volume fluxes across the 'u' and 'v' grid cell faces of a tracer cell could be calculated by multiplying the velocities at the face with the face area:

volume flux across 'u' face in the +x direction =
$UVEL(x,y,k) \times drF(k) \times dyG(x,y) \times hFacW(x,y,k)$

volume flux across 'v' face in the +y direction = $VVEL(x,y,k) \times drF(k) \times dxG(x,y) \times hFacS(x,y,k)$

With `dyG` and `dxG` being the lengths of the 'u' and 'v' faces, `drF` being the grid cell height and `hFacW` and `hFacS` being the vertical fractions of the 'u' and 'v' grid cell faces that are open water (ECCOv4 uses partial cells to better represent bathymetry which can allows 0 < hfac $\leq$ 1).

However, because the vertical coordiate varies with time in the z^* system, the grid cell height `drF` varies with time as `drF` $\times s^*(t)$, with

$$s^*(x,y,k,t) = 1 + \frac{\eta(x,y,t)}{H}$$

with $s^* > 1$ when $\eta > 0$

Thus, to calculate the volume fluxes grid cell through horizontal faces we must account for the time-varying grid cell face areas:

Evaluating the model sea level anomaly ETAN volume budget

We will evalute

$$\underbrace{\frac{\partial \eta}{\partial t}}_{G_{\text{total tendency}}} = \underbrace{\int_{-H}^{0} \left(-\nabla_{z^*} (s^* \mathbf{v} - \frac{\partial w}{\partial z^*}) \right) dz^*}_{G_{\text{volumetric divergence}}} + \underbrace{F}_{G_{surface fluxes}}$$

The total tendency of η, $G_{\text{total tendency}}$ is the sum of the η tendencies from volumetric divergence, $G_{\text{volumetric divergence}}$, and volumetric surface fluxes, $G_{\text{surface fluxes}}$.

In discrete form, using indexes that start from k=0 (surface tracer cell) and running to k=nk-1 (bottom tracer cell)

$$\begin{aligned}\frac{\eta(i,j)}{\partial t} = &\sum_{k=nk-1}^{0} \underbrace{[UVELMASS(i_g,j,k) - UVELMASS(i_g+1,j,k)]\, dyG(i_g,j)\, drF(k)}_{\text{volumetric flux in minus out in x direction}} + \\ &\sum_{k=nk-1}^{0} \underbrace{[VVELMASS(i,j_g,k) - VVELMASS(i,j_g+1,k)]\, dxG(i,j_g)\, drF(k)}_{\text{volumetric flux in minus out in y direction}} + \\ &\sum_{k_l=nk}^{1} \underbrace{WVELMASS(i,j,k_l)\, drA(i,j)}_{\text{volumetric flux through grid cell bottom surface}} + \\ &\underbrace{oceFWflx(i,j)/rhoConst}_{\text{volumetric flux through the top surface of the uppermost tracer cell}}\end{aligned}$$

In the above we intentionally sum `WVELMASS` fluxes from the BOTTOM surface of the lowermost grid cell (at $k_l = 50$) to the BOTTOM face of the uppermost grid cell ($k_l = 1$) so that we can explicitly include the surface volume flux (forcing) term, $oceFWflx(i,j)/rhoConst$.

We will calculate $\partial\eta/\partial t$ by differencing instantaneous monthly snapshots of η as

$$\frac{\partial \eta}{\partial t} = \frac{\eta(i,j,t+1) - \eta(i,j,t)}{\Delta t}$$

The `UVELMASS, VVELMASS, WVELMASS` and `oceFWflx` terms must be time-average quantities between the monthly η snapshots.

Open 2D MONTHLY η snapshots

```
year_start = 1993
year_end = 2016


# open ETAN snapshots (beginning of each month)
ecco_monthly_snaps_etan = (ds_dict[ShortNames_list[-2]]['ETAN']).to_dataset()
ecco_monthly_snaps_obp = ds_dict[ShortNames_list[-1]]
ecco_monthly_snaps = xr.merge((ecco_monthly_snaps_etan,ecco_monthly_snaps_obp))

# time mask for snapshots
time_snap_mask = np.logical_and(ecco_monthly_snaps.time.values >= np.datetime64(str(year_st
                                ecco_monthly_snaps.time.values < np.datetime64(str(year_end

ecco_monthly_snaps = ecco_monthly_snaps.isel(time=time_snap_mask)
```

```
# print time range of monthly snapshots
print(ecco_monthly_snaps.time.isel(time=[0, -1]).values)
```

```
['1993-01-01T00:00:00.000000000' '2017-01-01T00:00:00.000000000']
```

```
# find the record of the last ETAN snapshot
last_record_date = ecco_monthly_snaps.time[-1].values
print(last_record_date)
last_record_year = str(last_record_date)[:4]
```

```
2017-01-01T00:00:00.000000000
```

Open MONTHLY mean data

```
## Load ECCO variables
ecco_vars_uvw = ds_dict[ShortNames_list[1]]
ecco_vars_fflx = ds_dict[ShortNames_list[2]]
ecco_monthly_mean = xr.merge((ecco_vars_uvw,ecco_vars_fflx['oceFWflx']))

# time mask for monthly means
time_mean_mask = np.logical_and(ecco_monthly_mean.time.values >= np.datetime64(str(year_sta
                                ecco_monthly_mean.time.values < np.datetime64(str(year_end+

ecco_monthly_mean = ecco_monthly_mean.isel(time=time_mean_mask)
```

```
ecco_monthly_mean
```

```
# the weights are just the # of seconds per month divided by total seconds
month_length_weights = secs_per_month / secs_per_month.sum()
```

The time mean of the `ETAN` tendency, $\overline{G_{\text{total tendency}}}$, is given by

$$\overline{G_{\text{total tendency}}} = \sum_{i=1}^{nm} w_i G_{\text{total tendency}}$$

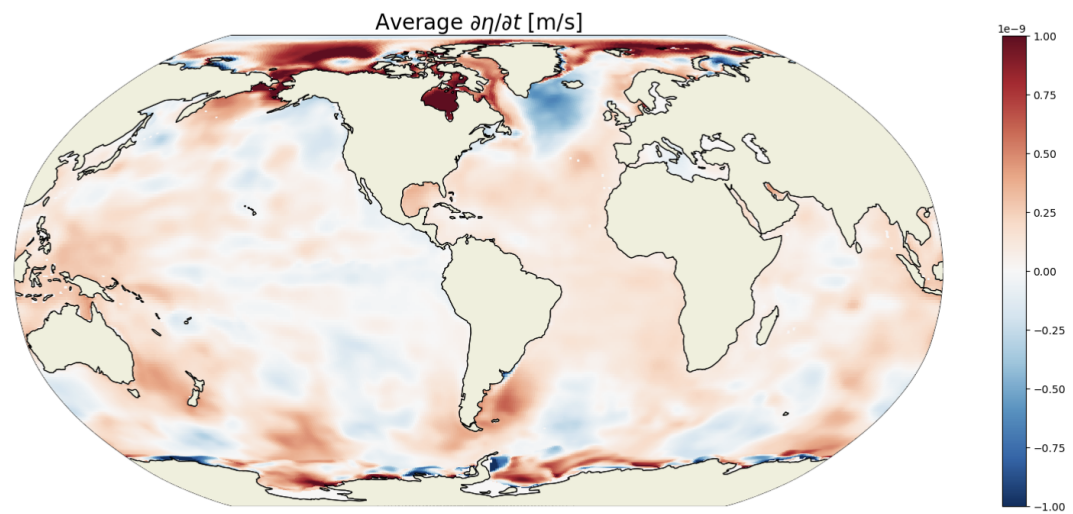
with $\sum_{i=1}^{nm} w_i = 1$ and nm=number of months

```
# the weights sum to 1
print(month_length_weights.sum())
```

```
<xarray.DataArray ()> Size: 8B
array(1.)
```

```
# the weighted mean weights by the length of each month (in seconds)
G_total_tendency_mean = (G_total_tendency*month_length_weights).sum('time')
```

```
plt.figure(figsize=(20,8));
ecco.plot_proj_to_latlon_grid(ecco_grid.XC, ecco_grid.YC,\
                              G_total_tendency_mean,show_colorbar=True,\
                              cmin=-1e-9, cmax=1e-9, \
                              cmap='RdBu_r', user_lon_0=-67,\
                              dx=map_dx,dy=map_dy);
plt.title('Average $\partial \eta / \partial t$ [m/s]', fontsize=20);
```



Gradients and curls of vector quantities