oocgcm Documentation

Release 0.1.0.dev-c961687

oocgcm developpers

January 09, 2017

Contents

This project provides tools for processing and analysing output of general circulation models and gridded satellite data in the field of Earth system science.

Our aim is to simplify the analysis of **very large datasets of model output** (~1-100Tb) like those produced by basin-to-global scale sub-mesoscale permitting ocean models and ensemble simulations of eddying ocean models by **leveraging the potential of** xarray **and** dask python packages.

The main ambition of this project is to provide simple tools for performing **out-of-core** computations with model output and gridded data, namely processing data that is too large to fit into memory at one time.

The project is so far mostly targeting NEMO ocean model and gridded ocean satellite data (AVISO, SST, ocean color...) but our aim is to build **a framework that can be used for a variety of models based on the Arakawa C-grid**. The framework can in principle also be used for atmospheric general circulation models.

We are trying to develop a framework **flexible** enough in order not to impose too strictly a specific workflow to the end user.

oocgcm is a pure Python package and we try to keep the **list of dependencies** as small as possible in order to simplify the deployment on a number of platforms.

oocgcm is **not intended to provide advanced visualization** functionalities for gridded geographical data as several powerful tools already exist in the python ecosystem (see in particular cartopy and basemap).

We rather focus on building a framework that simplifies the design and production of **advanced statistical and dynamical analyses of large datasets** of model output and gridded data.

Note: oocgcm is at the pre-alpha stage. The API is therefore unstable and likely to change without notice.

CHAPTER 1

Documentation

1.1 The need for out-of-core analysis tools

1.1.1 A practical need

oocgcm is a pure Python package built on the top of xarray and dask.

Most of the tools that are implemented in oocgcm are already available in several python libraries dedicated to the analysis of gridded data, and based on numpy and one of the several netCDF interfaces for python.

Why have we choosen to implement a diagnostic package based on xarray and dask ?

Numpy-based model diagnostic libraries are facing a challenge with the ongoing evolution of geoscientific models and earth observing networks. With **the most high-end models being runs on several tens of thousand cores**, even a two-dimensional slice of model output cannot be loaded in memory at one time. Model diagnostic tools and gridded data analysis **tools should therefore be parallelized** and run out-of-core.

One option is to run FORTRAN/MPI codes steered from shell scripts but there you loose the **flexibility and the multiple benefits of a python-based workflow**. Another option is to use one of the several libraries available for parallel computing in python. This usually implies a complete refactoring of your analysis package at the risk of eventually using different analysis tools depending on the size of your dataset...

1.1.2 A wider perspective

We believe that the above question is not a "technical" problem but **a real challenge for our fields of research**. We are here facing a "technical" translation of one of the "big data challenges" in earth system science.

Transforming large gridded datasets into scientific results indeed requires **innovative descriptive approaches that merge statistical descriptions and physically-motivated analyses**. This usually involves performing rather "complex" analysis tasks on the dataset.

In practice, this is usually a two stage process. A first stage involves transforming an **idea** into a **prototype** code. Using a interactive scripting language usually accelerates this phase because you can glue different bits of code and see the results on the fly.

The second stage involves transforming the **prototype** code into a **production** code. And this is usually a difficult task especially for students, because it requires a more in-depth understanding of the hardware infrastructure and of the software design.

So, depending on the language and on the libraries you use, what actually changes is not your ability to perform your analysis but rather **the time it takes for you to prepare your production code** and eventually reach the scientific result you are after.

More generally, there is an objective risk for our fields of research if we don't embrace the question of the development of analysis tools that **accelerate the idea-prototype-production-results cycle**. If the time needed to transform an idea into an efficient production code is too long, we will keep performing only simple or routine analyses on our datasets, eventually missing **the potential breakthrough of big-data in earth system sciences**.

1.2 Why using xarray and dask ?

oocgcm is a pure Python package built on the top of xarray, which itself integrates dask.array to support streaming computation on large datasets that don't fit into memory. Why have we choosen to use xarray and dask ?

1.2.1 xarray

xarray implements a N-dimensional variants of the core pandas data structures. In addition, xarray adopts the Common Data Model for self-describing data. In practice, xarray.Dataset is an in-memory representation of a netCDF file or of a collection of netCDF files.

Building upon xarray has several advantages :

- metadata available in the netCDF files are associated with xarray objects in the form of a Python dictionary `x.attrs`. This simplifies the exploration of the dataset, yields more robust code, and simplifies the export of the results to netCDF files.
- because dimensions are associated with the variable in xarray objects, xarray allows flexible split-apply-combine operations with groupby `x.groupby('time.dayofyear').mean()`
- xarray objects do not load data in memory by default. Loading the data is only done at the execution time if needed. This means that the user has access to all his dataset without having to worry about loading the data, therefore simplifying the prototyping of a new analysis.
- xarray is natively integrated with pandas, meaning that xarray objects can straightforwardly be exported to pandas DataFrames. This allows to easily access a range of time-series analysis tools.
- xarray objects can be exported to iris or cdms so that the user can merge several different analysis tools in his workflow.
- Little work is needed for applying a numpy function to xarray objects. Several numpy ufunc are already applicable to xarray.DataArray data-structure.

1.2.2 dask

dask implement an abstract graph representation of the dynamic task scheduling needed for performing out-of-core computation. dask also implement an efficient scheduling procedure for optimizing the execution time of acyclic graphs (DAG) of tasks on a given machine.

From a user standpoint the key concept of dask.array is the notion of chunk. A chunk is the user-defined shape of the subdataset on which the unitary tasks will be applied.

dask allows to easily leverage the resources of **shared memory architectures** (multi-core laptop or work-station) but also the resources of **distributed memory architectures** (clusters of cpu).

At present, xarray integrates dask functionalities for shared memory architectures. xarray will also allow to leverage dask potential on distributed memory architectures in the future.

Building upon dask has several advantages :

- parallelization comes at no cost. The only modification of your code that is needed is your defining the chunks on which the computation should be performed.

- dask back-end methods are generic, powerful and well tested for a number of different applications.
- dask comes with powerful and easy-to-use profiling tools for optimizing the execution time on a given machine.

Most importantly, xarray and dask are supported by active and friendly teams of developers, that we hereby gratefully acknowledge.

1.3 Installation

1.3.1 Dependancies

oocgcm is a pure Python package, but some of its dependencies are not. The easiest way to get them installed is to use conda. You can install all oocgcm dependancies with the conda command line tool:

```
conda install xarray dask netCDF4 bottleneck
```

1.3.2 Installing

The project is still in pre-alpha phase, the only way to install oocgcm is therefore to clone the repository on github:

```
git clone https://github.com/lesommer/oocgcm.git
cd oocgcm
python setup.py install
```

1.3.3 Test

Test oocgcm with `py.test`:

```
cd oocgcm
py.test oocgcm
```

1.4 Generic versus data-specific tools

oocgcm is designed so that most methods can be used with several different types of gridded data.

There are two classes of methods in oocgcm. Methods that are specific to a particular model or source of data and methods that are data-agnostic or generic.

For generic tools, we try to use naming conventions for internal variables based on CF conventions or COMODO conventions.

Data-specific methods are generally created as particular instances of generic methods. This is in particular the case for grid descriptor objects.

The distinction between data-agnostic and data-specific methods is reflected in the general structure of the code.

All the methods related to a particular sources of data are put together in a specific folder (eg. oocgcm.oceanmodels.nemo) so that it is easier for third party to maintain data-specific tools associated to their particular source of data / model.

1.5 Adapted to different usage

We would like oocgcm to be generic enough not to impose to strictly a particular workflow to the end user.

oocgcm can be used interactively in ipython or jupyter notebooks.

oocgcm can also be used for building individual python scripts that read and write collections of netcdf files (a la nco).

1.6 Grid descriptor objects

One of the key concepts of oocgcm is the notion of grid that is implemented in py:module:*oocgcm.core.grids*. The current version implements a generic two-dimensional lat/lon grid in py:class:*oocgcm.core.grids.generic_2d_grid*. Future versions will have a similar object for three-dimensional data.

Grid objects can be created in various ways depending on the source of gridded data. A present, two-dimensional grid objects can be created :

- from arrays of latitude and longitude,
- from arrays of projection coordinate x and y,
- from NEMO ocean model netCDF output files.

Grid descriptors provide access to all the information that may be needed for defining operations on the grid. They implement methods for vector calculus, differential calculus and spatial integration. Grid descriptors can also be sliced for defining descriptors of smaller portions of the physical domain.

In practice, a grid object can be created from arrays of latitude and longitude (1d or 2d arrays).

```
from oocgcm.griddeddata import grids
lats = ...
lons = ...
grd = grids.latlon_2d_grid(latitudes=lats,longitudes=lons)
```

Grids descriptors can also be constructed from netCDF files describing the model grid (only available for NEMO ocean model so far).

```
from oocgcm.oceanmodels.nemo import grids
grd = grids.nemo_2d_grid(nemo_coordinate_file=...,nemo_byte_mask_file=...,chunks=...)
```

Note: Construction from netCDF files should be preferred as the metric factors that described the grid (e.g. `grd["cell_x_size_at_t_location"]`) are more accurate in this case.

Grids descriptors can also be constructed from x,y coordinate (in m). This can be useful for analysing idealized model experiments (eg. on the f-plane or beta-plane).

```
from oocgcm.griddeddata import grids
x = np.arange(start=0, stop=1.e7, step=1.e6,dtype=float)
y = np.arange(start=0, stop=1.2e7, step=1.e6,dtype=float)
grd = grids.plane_2d_grid(ycoord=y,xcoord=x)
```

Note: It should be noted that the creation of grid descriptors does not load any data nor creates any additional numpy arrays. A grid descriptors only defines xarray.DataArray instances. Grid object just contain the information on how to perform a particular calculation.

Once created, a grid descriptor `grd` can be restricted to a subdomain as follows

```
zoom = grd[500:800,2000:2500]
```

1.6.1 Operations on vector fields

Grid descriptor objects allow to perform **operations on vector fields** (as defined in oocgcm.core). For instance,

```
from oocgcm.core.grids import VectorField2d
vectorfield1 = VectorField2d(...,...)
vectorfield2 = VectorField2d(...,...)
sprod = grd.scalar_product(vectorfield1,vectorfield2)
cprod = grd.vertical_component_of_the_cross_product(vectorfield1,vectorfield2)
```

Note: Methods associated to grid object only define xarray.DataArray instances. The actual computation only occurs when xarray.DataArray values are explicitely requested.

1.6.2 Differential operators

Grid descriptor objects implement **differential operators** that can be applied to gridded data :

```
sst = ...
u = ...
v = ...
current = VectorField2d(u,v)
gsst = grd.horizontal_gradient(sst)
curl = grd.vertical_component_of_curl(current)
```

Note that by default, oocgcm uses low order discretization methods. This can be easily overridden by the user provided the reference API is followed.

1.6.3 Spatial integration

Grid descriptors also implement methods for performing spatial integration :

```
ssh = ...
sst = ...
index = grd.spatial_average_xy(ssh,where=sst<2.) # average ssh where sst <2.
index.plot() # plots the time series
```

1.6.4 Broadcasting additional dimensions

Methods associated with two-dimensional grid descriptors define operations based on ('y','x') dimensions. Because of the flexibility of xarray, these methods can therefore be applied to dataarrays with additional extra dimensions (for instance, time, index in an ensemble simulation).

For instance, if `ssh` is a xarray dataarray of sea surface height built from a collection of netCDF files corresponding to different times,

```
import xarray as xr
xr.open_mfdataset(path_to_my_files)['sossheig']
```

the norm of sea surface heigh gradient is defined as follows

```
gssh = grd.norm_of_vectorfield(grd.horizontal_gradient(ssh))
```

but `gssh` does not hold any data yet. The calculation is only performed when a partical slice of data is requested, as for instance if the first two-dimensional field is written in a netCDF file.

```
gssh[0].to_netcdf(path_to_my_output_file)
```

This abstract representation of operations that allows xarray is key for efficiently implementing out-of-core procedures. It also allows to straighforwardly deal **with grids with time-varying metrics**. This is key for working with arbitrary **lagrangian-eulerian vertical coordinates**.

1.7 Tools for simplifying I/O

Currently oocgcm provides tools that simplify the creation of xarray.DataArray from a physical database (structured collection of netcdf files).

oocgcm will soon provide tools that facilitate the creation of physical database resulting from analysis performed with oocgcm, following particular conventions (eg for instance DRAKKAR conventions).

1.8 API reference

This page provides an auto-generated summary of oocgcm's API.

1.8.1 General purpose utilities and data-structures

Parameters

parameters.physicalparameters.coriolis_parameter(...)	Return Coriolis parameter for a given latitude.
parameters.physicalparameters.beta_parameter(...)	Return planetary beta parameter.

oocgcm.parameters.physicalparameters.coriolis_parameter

oocgcm.parameters.physicalparameters.**coriolis_parameter**(*latitudes*)

Return Coriolis parameter for a given latitude.

Parameters **latitudes** : float or array-like

latitudes can be a float, a numpy array or a xarray.

Returns **corio** : same type as input

oocgcm.parameters.physicalparameters.beta_parameter

oocgcm.parameters.physicalparameters.**beta_parameter**(*latitudes*)

Return planetary beta parameter.

Parameters **latitudes** : float or array-like

latitudes can be a float, a numpy array or a xarray.

Returns **beta** : float of array-like

Data-structures

core.grids.VectorField2d(vx, vy[, ...])	Minimal data structure for manupulating 2d vector fields on a grid.
core.grids.VectorField3d(vx, vy, vz[, ...])	Minimal data structure for manupulating 3d vector fields on a grid.
core.grids.Tensor2d(axx, axy, ayx, ayy[, ...])	Minimal data structure for manupulating 2d tensors on a grid.

oocgcm.core.grids.VectorField2d

oocgcm.core.grids.**VectorField2d**(*vx*, *vy*, *x_component_grid_location=None*, *y_component_grid_location=None*)

Minimal data structure for manupulating 2d vector fields on a grid.

Parameters **vx** : xarray.DataArray

x-component of the vector fields

vy : xarray.DataArray

y-component of the vector fields

x_component_grid_location : str

string describing the grid location of the x-component

y_component_grid_location : str

string describing the grid location of the y-component

Returns **o** : namedtuple

namedtuple containing the vector field.

oocgcm.core.grids.VectorField3d

oocgcm.core.grids.**VectorField3d**(*vx*, *vy*, *vz*, *x_component_grid_location=None*, *y_component_grid_location=None*, *z_component_grid_location=None*)

Minimal data structure for manupulating 3d vector fields on a grid.

Parameters **vx** : xarray.DataArray

x-component of the vector fields

vy : xarray.DataArray

y-component of the vector fields

vz : xarray.DataArray

z-component of the vector fields

x_component_grid_location : str

string describing the grid location of the x-component

y_component_grid_location : str

string describing the grid location of the y-component

z_component_grid_location : str

string describing the grid location of the z-component

Returns **o** : namedtuple

namedtuple containing the vector field.

oocgcm.core.grids.Tensor2d

oocgcm.core.grids.**Tensor2d**(*axx*, *axy*, *ayx*, *ayy*, *xx_component_grid_location=None*, *xy_component_grid_location=None*, *yx_component_grid_location=None*, *yy_component_grid_location=None*)

Minimal data structure for manupulating 2d tensors on a grid.

Parameters **axx** : xarray.DataArray

xx-component of the tensor

axy : xarray.DataArray

xy-component of the tensor

ayx : xarray.DataArray

yx-component of the tensor

ayy : xarray.DataArray

yy-component of the tensor

xx_component_grid_location : str

string describing the grid location of the xx-component

xy_component_grid_location : str

string describing the grid location of the xy-component

yx_component_grid_location : str

string describing the grid location of the yx-component

yy_component_grid_location : str

string describing the grid location of the yy-component

Returns **o** : namedtuple

namedtuple containing the tensor.

Notes

uses the following notations : .. math:

```
\mathbf{T} =
egin{pmatrix}
a_{xx} & a_{xy} \
a_{yv} & a_{yy}
\end{pmatrix}
```

Continued on next pag

Table 1.3 – continued from previous page

I/O tools

core.io.return_xarray_dataset(filename[, chunks])	Return an xarray dataset corresponding to filename.
core.io.return_xarray_dataarray(filename, ...)	Return a xarray dataarray corresponding to varname in filenam
oceanmodels.nemo.io.return_xarray_dataset(...)	Wrapper for core.io.return_xarray_dataset.
oceanmodels.nemo.io.return_xarray_dataarray(...)	Wrapper for core.io.return_xarray_dataarray.

oocgcm.core.io.return_xarray_dataset

oocgcm.core.io.**return_xarray_dataset**(*filename*, *chunks=None*, ***kwargs*)

Return an xarray dataset corresponding to filename.

Parameters **filename** : str

path to the netcdf file from which to create a xarray dataset

chunks : dict-like

dictionnary of sizes of chunk for creating xarray.Dataset.

Returns **ds** : xarray.Dataset

oocgcm.core.io.return_xarray_dataarray

oocgcm.core.io.**return_xarray_dataarray**(*filename*, *varname*, *chunks=None*, ***extra_kwargs*)

Return a xarray dataarray corresponding to varname in filename.

Parameters **filename** : str

path to the netcdf file from which to create a xarray.DataArray

varname : str

name of the variable from which to create a xarray.DataArray

chunks : dict-like

dictionnary of sizes of chunk for creating a xarray.DataArray.

****extra_kwargs**

not used

Returns **da** : xarray.DataArray

oocgcm.oceanmodels.nemo.io.return_xarray_dataset

oocgcm.oceanmodels.nemo.io.**return_xarray_dataset**(**args*, ***kwargs*)

Wrapper for core.io.return_xarray_dataset.

Parameters ***args** : arguments

****kwargs** : keyword arguments

Returns

———

ds : xarray.Dataset

Methods

——

change the name of dimension 'time_counter' in 't'

oocgcm.oceanmodels.nemo.io.return_xarray_dataarray

oocgcm.oceanmodels.nemo.io.**return_xarray_dataarray**(**args*, ***kwargs*)

Wrapper for core.io.return_xarray_dataarray.

Parameters ***args** : arguments

****kwargs** : keyword arguments

Returns **da** : xarray.DataArray

Methods

change the name of dimension 'time_counter' in 't'	

Miscellaneous

core.utils.is_numpy(array)	Return True if array is a numpy array
core.utils.is_xarray(array)	Return True if array is a xarray.DataArray
core.utils.is_daskarray(array)	Return True if array is a dask array
core.utils.has_chunks(array)	Return True if array is a xarray or a daskarray with chunks.
core.utils.map_apply(func, scalararray)	Return a xarray dataarray with value func(scalararray.data)

oocgcm.core.utils.is_numpy

oocgcm.core.utils.**is_numpy**(*array*)

Return True if array is a numpy array

Parameters **array** : array-like

array is either a numpy array, a masked array, a dask array or a xarray.

Returns **test** : bool

oocgcm.core.utils.is_xarray

oocgcm.core.utils.**is_xarray**(*array*)

Return True if array is a xarray.DataArray

Parameters **array** : array-like

Returns **test** : bool

oocgcm.core.utils.is_daskarray

oocgcm.core.utils.**is_daskarray**(*array*)
Return True if array is a dask array

Parameters **array** : array-like

Returns **test** : bool

oocgcm.core.utils.has_chunks

oocgcm.core.utils.**has_chunks**(*array*)
Return True if array is a xarray or a daskarray with chunks.

Parameters **array** : array-like

array is either a numpy array, a masked array, a dask array or a xarray.

Returns **test** : bool

oocgcm.core.utils.map_apply

oocgcm.core.utils.**map_apply**(*func*, *scalararray*)
Return a xarray dataarray with value func(scalararray.data)

Parameters **func** : function

Any function that works on numpy arrays such that input and output arrays have the same shape.

scalararray : xarray.DataArray

Returns **out** : xarray.DataArray

Methods

uses dask map_block without ghost cells (map_overlap)	

1.8.2 Two-dimensional grid descriptor objects

Two-dimensional grid descriptors hold all the information required for defining operations on xarray dataarray that require information on the underlying grid of the model.

Generic two-dimensional model-agnostic grid descriptor

core.grids.generic_2d_grid([arrays, parameters])	Model agnostic grid object, two dimensional version.

oocgcm.core.grids.generic_2d_grid

class oocgcm.core.grids.**generic_2d_grid**(*arrays=None*, *parameters=None*)
Model agnostic grid object, two dimensional version.

This class holds the xarrays that describe the grid and implements grid related methods.

This includes :

- vector calculus (scalar product, norm, vector product,...)
- interpolation between different grid locations (eg. 'u'–>'t')
- differential operators (gradient, divergence, etc...)
- spatial integration

Most methods expect and return instance of xarray.DataArray.

Assume that dimension names are 'x' and 'y'.

__init__(*arrays=None*, *parameters=None*)

Initialize a grid from a dictionary of xarrays and some parameters.

Parameters **variables** : dict-like object

dictionnary of xarrays that describe the grid. Required variables for the method actually implemented are listed in _required_arrays. This naming convention follows a mixture of cf and comodo norms in order for this class to be model-agnostic.

parameters : dict-object

not used yet.

Methods

__init__([arrays, parameters])	Initialize a grid from a dictionary of xarrays and some paramet
change_grid_location_f_to_u(scalararray[, ...])	Return a xarray corresponding to scalararray averaged at a new
change_grid_location_f_to_v(scalararray[, ...])	Return a xarray corresponding to scalararray averaged at a new
change_grid_location_t_to_u(scalararray[, ...])	Return a xarray corresponding to scalararray averaged at a new
change_grid_location_t_to_v(scalararray[, ...])	Return a xarray corresponding to scalararray averaged at a new
change_grid_location_u_to_t(scalararray[, ...])	Return a xarray corresponding to scalararray averaged at a new
change_grid_location_u_to_v(scalararray[, ...])	Return a xarray corresponding to scalararray averaged at a new
change_grid_location_v_to_t(scalararray[, ...])	Return a xarray corresponding to scalararray averaged at a new
change_grid_location_v_to_u(scalararray[, ...])	Return a xarray corresponding to scalararray averaged at a new
chunk([chunks])	Rechunk all the variables defining the grid.
frontogenesis_function(velocity, buoyancy)	Return the frontogenesis function.
geostrophic_current_from_sea_surface_height(...)	Return the geostrophic current on u,v-grids.
get_projection_coordinates([grid_location])	Return (x,y) the coordinate arrays (in m) at grid location.
horizontal_divergence(vectorfield)	Return the horizontal divergence of a vector field at u,v-points.
horizontal_gradient(datastructure)	Return the horizontal gradient of the input datastructure.
horizontal_gradient_tensor(vectorfield)	Return the horizontal gradient tensor of a two-dimensional vect
horizontal_gradient_vector(scalararray)	Return the horizontal gradient of a scalar field defined at t-poin
horizontal_laplacian(scalararray)	Return the horizontal laplacian of a scalar field at t-points.
integrate_dxdy(array[, where, ...])	Return the horizontal integral of array in regions where where i
norm_of_vectorfield(vectorfield)	Return the norm of a vector field, at t-point.
q_vector_due_to_kinematic_deformation(...)	Return the component of the generalized Q-vector associated w
scalar_outer_product(scalararray, vectorfield)	Return the outer product of a scalar (t location)
scalar_product(vectorfield1, vectorfield2)	Return the scalar product of two vector fields, at t-point.
spatial_average_xy(array[, where, grid_location])	Return the horizontal average of array in regions where where i
vertical_component_of_curl(vectorfield)	Return the vertical component of the curl of a vector field.
vertical_component_of_the_cross_product(...)	Return the cross product of two vector fields.

Attributes

chunks	Chunks of the xarray dataarrays describing the grid.
dims	Dimensions of the xarray dataarrays describing the grid.
ndims	Number of dimensions of the dataarrays describing the grid.
shape	Shape of the xarray dataarrays describing the grid.

Creating grids descriptor from arrays

griddeddata.grids.variables_holder_for_2d_grid_from_latlon_arrays([...])	This class cre
griddeddata.grids.latlon_2d_grid([...])	Return a gene
griddeddata.grids.variables_holder_for_2d_grid_from_plane_coordinate_arrays([...])	This class cre
griddeddata.grids.plane_2d_grid([xcoord, ...])	Return a gene

oocgcm.griddeddata.grids.variables_holder_for_2d_grid_from_latlon_arrays

class oocgcm.griddeddata.grids.**variables_holder_for_2d_grid_from_latlon_arrays**(*latitudes=None*, *longitudes=None*, *mask=None*, *chunks=None*)

This class create the dictionnary of variables to be used for creating a oocgcm.core.grids.generic_2d_grid from arrays of latitude and longitude.

__init__(*latitudes=None*, *longitudes=None*, *mask=None*, *chunks=None*)

This holder will create grid metrics and masks corresponding to a Arakawa C-grid assuming that latitudes, longitudes and mask refer to the centers of the cells (t-points).

Parameters **latitudes** : array-like (numpy array or xarray)

array of latitudes from which to build the grid metrics. This can be either a one-dimensionnal or a two-dimensional array. For 2d arrays, assume that the order of the dimensions is ('y','x').

longitudes : array-like (numpy array or xarray)

array of longitudes from which to build the grid metrics. This can be either a one-dimensionnal or a two-dimensional array. For 2d arrays, assume that the order of the dimensions is ('y','x').

mask : boolean array-like (numpy array or xarray)

two-dimensional array describing the inner domain of the grid. mask==1 within the ocean, mask==0 on land.

chunks : dict-like

dictionnary of sizes of chunk for creating xarray.DataArray.

Methods

__init__([latitudes, longitudes, mask, chunks])	This holder will create grid metrics and masks corresponding to a Arakawa C-gri

oocgcm.griddeddata.grids.latlon_2d_grid

oocgcm.griddeddata.grids.**latlon_2d_grid**(*latitudes=None*, *longitudes=None*, *mask=None*, *chunks=None*)

Return a generic 2d grid build from arrays of lat,lon.

The grid object corresponds to a Arakawa C-grid assuming that latitudes, longitudes and mask refer to the centers of the cells.

Parameters **latitudes** : array-like (numpy array or xarray)

array of latitudes from which to build the grid metrics. This can be either a one-dimensionnal or a two-dimensional array. For 2d arrays, assume that the order of the dimensions is ('y','x').

longitudes : array-like (numpy array or xarray)

array of longitudes from which to build the grid metrics. This can be either a one-dimensionnal or a two-dimensional array. For 2d arrays, assume that the order of the dimensions is ('y','x').

mask : boolean array-like (numpy array or xarray)

two-dimensional array describing the inner domain of the grid. mask==1 within the ocean, mask==0 on land.

chunks : dict-like

dictionnary of sizes of chunk for creating xarray.DataArray.

Returns **grid** : oocgcm.core.grids.generic_2d_grid

grid object corresponding to the input lat,lon arrays.

oocgcm.griddeddata.grids.variables_holder_for_2d_grid_from_plane_coordinate_arrays

class oocgcm.griddeddata.grids.**variables_holder_for_2d_grid_from_plane_coordinate_arrays**(*xcoor*
yco-
ord=
mask
chunk
lat=4
lon=0)

This class create the dictionnary of variables to be used for creating a oocgcm.core.grids.generic_2d_grid from arrays of plane coordinates.

__init__(*xcoord=None*, *ycoord=None*, *mask=None*, *chunks=None*, *lat=45*, *lon=0*)

This holder will create grid metrics and masks corresponding to a Arakawa C-grid assuming that xcoord, ycoord and mask refer to the centers of the cells (t-points). This class is meant to be used for idealized domains (eg.g on the f-plane of beta-plane)

Parameters **xcoord** : array-like (numpy array or xarray)

array of x-coordinate from which to build the grid metrics. This can be either a one-dimensionnal or a two-dimensional array. For 2d arrays, assume that the order of the dimensions is ('y','x').

ycoord : array-like (numpy array or xarray)

array of y-coordinate from which to build the grid metrics. This can be either a one-dimensionnal or a two-dimensional array. For 2d arrays, assume that the order of the dimensions is ('y','x').

mask : boolean array-like (numpy array or xarray)

two-dimensional array describing the inner domain of the grid. mask==1 within the ocean, mask==0 on land.

chunks : dict-like

dictionnary of sizes of chunk for creating xarray.DataArray.

lat : float or array-like

latitude of t-points. This can be a float, a 1d or a 2d numpy array.

lon : float or array-like

longitude of t-points. This can be a float, a 1d or a 2d numpy array.

Methods

`__init__`([xcoord, ycoord, mask, chunks, ...])	This holder will create grid metrics and masks corresponding to a Arakawa C-grid

oocgcm.griddeddata.grids.plane_2d_grid

oocgcm.griddeddata.grids.**plane_2d_grid**(*xcoord=None*, *ycoord=None*, *mask=None*, *chunks=None*, *lat=45.0*, *lon=0.0*)

Return a generic 2d grid build from arrays of coordinate.

The grid object corresponds to a Arakawa C-grid assuming that xcoord, ycoord and mask refer to the centers of the cells.

Parameters **xcoord** : array-like (numpy array or xarray)

array of x-coordinate from which to build the grid metrics. This can be either a one-dimensionnal or a two-dimensional array. For 2d arrays, assume that the order of the dimensions is ('y','x').

ycoord : array-like (numpy array or xarray)

array of y-coordinate from which to build the grid metrics. This can be either a one-dimensionnal or a two-dimensional array. For 2d arrays, assume that the order of the dimensions is ('y','x').

mask : boolean array-like (numpy array or xarray)

two-dimensional array describing the inner domain of the grid. mask==1 within the ocean, mask==0 on land.

chunks : dict-like

dictionnary of sizes of chunk for creating xarray.DataArray.

lat : float or array-like

latitude of t-points. This can be a float, a 1d or a 2d numpy array.

lon : float or array-like

longitude of t-points. This can be a float, a 1d or a 2d numpy array.

Returns **grid** : oocgcm.core.grids.generic_2d_grid

grid object corresponding to the input xcoord,ycoord arrays.

Creating grids descriptors from model output

oceanmodels.nemo.grids.variables_holder_for_2d_grid_from_nemo_ogcm([...])	This class create the diction
oceanmodels.nemo.grids.nemo_2d_grid([...])	Return a generic 2d grid fr

oocgcm.oceanmodels.nemo.grids.variables_holder_for_2d_grid_from_nemo_ogcm

class oocgcm.oceanmodels.nemo.grids.**variables_holder_for_2d_grid_from_nemo_ogcm**(*nemo_coordinate_f*
nemo_byte_mask_f
chunks=None,
byte_mask_level=0)

This class create the dictionnary of variables used for creating a oocgcm.core.grids.generic_2d_grid from NEMO output files.

__init__(*nemo_coordinate_file=None*, *nemo_byte_mask_file=None*, *chunks=None*, *byte_mask_level=0*)

This holder uses the files meshhgr.nc and byte_mask.nc

Parameters **nemo_coordinate_file** : str

path to NEMO coordinate file associated to the model configuration.

nemo_byte_mask_file : str

path to NEMO mask file associated to the model configuration.

chunks : dict-like

dictionnary of sizes of chunk for creating xarray.DataArray.

byte_mask_level : int

index of the level from which the masks should be loaded

Methods

__init__([nemo_coordinate_file, ...])	This holder uses the files meshhgr.nc and byte_mask.nc
chunk([chunks])	Chunk all the variables.

oocgcm.oceanmodels.nemo.grids.nemo_2d_grid

oocgcm.oceanmodels.nemo.grids.**nemo_2d_grid**(*nemo_coordinate_file=None*,
nemo_byte_mask_file=None, *chunks=None*,
byte_mask_level=0)

Return a generic 2d grid from nemo coordinate and mask files.

Parameters **nemo_coordinate_file** : str

path to NEMO coordinate file associated to the model configuration.

nemo_byte_mask_file : str

path to NEMO mask file associated to the model configuration.

chunks : dict-like

dictionnary of sizes of chunk for creating xarray.DataArray.

byte_mask_level : int

index of the level from which the masks should be loaded

Returns **grid** : oocgcm.core.grids.generic_2d_grid

grid object corresponding to the model configuration.

Attributes

core.grids.generic_2d_grid.dims	Dimensions of the xarray dataarrays describing the grid.
core.grids.generic_2d_grid.ndims	Number of dimensions of the dataarrays describing the grid.
core.grids.generic_2d_grid.shape	Shape of the xarray dataarrays describing the grid.
core.grids.generic_2d_grid.chunks	Chunks of the xarray dataarrays describing the grid.

oocgcm.core.grids.generic_2d_grid.dims

generic_2d_grid.**dims**
Dimensions of the xarray dataarrays describing the grid.

oocgcm.core.grids.generic_2d_grid.ndims

generic_2d_grid.**ndims**
Number of dimensions of the dataarrays describing the grid.

oocgcm.core.grids.generic_2d_grid.shape

generic_2d_grid.**shape**
Shape of the xarray dataarrays describing the grid.

oocgcm.core.grids.generic_2d_grid.chunks

generic_2d_grid.**chunks**
Chunks of the xarray dataarrays describing the grid.

Manipulating grids

generic_2d_grid implement the mapping interface with keys given by variable names and values given by DataArray objects.

core.grids.generic_2d_grid.__getitem__(item)	The behavior of this function depends on the t
core.grids.generic_2d_grid.__contains__(key)	The 'in' operator will return true or false depe
core.grids.generic_2d_grid.__iter__()	
core.grids.generic_2d_grid.chunk([chunks])	Rechunk all the variables defining the grid.
core.grids.generic_2d_grid.get_projection_coordinates([...])	Return (x,y) the coordinate arrays (in m) at gri

oocgcm.core.grids.generic_2d_grid.__getitem__

generic_2d_grid.**__getitem__**(*item*)

The behavior of this function depends on the type of item.

- if item is a string, return the array self._arrays[item]
- if item is a slice, a grid object restricted to a subdomain.

Use slicing with caution, this functionnality depends on the order of the dimensions in the netcdf files.

Parameters **item** : slice or str

item can be a string identifying a key in self._arrays item can be a slice for restricting the grid to a subdomain.

Returns **out** : xarray.DataArray or generic_2d_grid

either a dataarray corresponding to self._arrays[item] or a new grid object corresponding to the restructed region.

oocgcm.core.grids.generic_2d_grid.__contains__

generic_2d_grid.**__contains__**(*key*)

The 'in' operator will return true or false depending on whether 'key' is an array in the dataset or not.

oocgcm.core.grids.generic_2d_grid.__iter__

generic_2d_grid.**__iter__**()

oocgcm.core.grids.generic_2d_grid.chunk

generic_2d_grid.**chunk**(*chunks=None*)

Rechunk all the variables defining the grid.

Parameters **chunks** : dict-like

dictionnary of sizes of chunk along xarray dimensions.

oocgcm.core.grids.generic_2d_grid.get_projection_coordinates

generic_2d_grid.**get_projection_coordinates**(*grid_location='t'*)

Return (x,y) the coordinate arrays (in m) at grid location.

Caution : This function may change in future versions of oocgcm.

Changing the grid location of arrays

core.grids.generic_2d_grid.change_grid_location_t_to_u(...)	Return a xarray corresponding to scalararray a
core.grids.generic_2d_grid.change_grid_location_u_to_t(...)	Return a xarray corresponding to scalararray a
core.grids.generic_2d_grid.change_grid_location_t_to_v(...)	Return a xarray corresponding to scalararray a
core.grids.generic_2d_grid.change_grid_location_v_to_t(...)	Return a xarray corresponding to scalararray a

Table 1.15 – continued from previous page

core.grids.generic_2d_grid.change_grid_location_f_to_u(...)	Return a xarray corresponding to scalararray a
core.grids.generic_2d_grid.change_grid_location_f_to_v(...)	Return a xarray corresponding to scalararray a
core.grids.generic_2d_grid.change_grid_location_v_to_u(...)	Return a xarray corresponding to scalararray a
core.grids.generic_2d_grid.change_grid_location_u_to_v(...)	Return a xarray corresponding to scalararray a

oocgcm.core.grids.generic_2d_grid.change_grid_location_t_to_u

generic_2d_grid.**change_grid_location_t_to_u**(*scalararray*, *conserving='area'*)

Return a xarray corresponding to scalararray averaged at a new grid location.

Parameters **scalararray** : xarray.DataArray

original array to be relocated

conserving : str

any of 'area', 'x_flux' or 'y_flux'.

- 'area' : conserves the area
- 'x_flux' : conserves the flux in x-direction (eastward)
- 'y_flux' : conserves the flux in y-direction (northward)

oocgcm.core.grids.generic_2d_grid.change_grid_location_u_to_t

generic_2d_grid.**change_grid_location_u_to_t**(*scalararray*, *conserving='x_flux'*)

Return a xarray corresponding to scalararray averaged at a new grid location.

Parameters **scalararray** : xarray.DataArray

original array to be relocated

conserving : str

any of 'area', 'x_flux' or 'y_flux'.

- 'area' : conserves the area
- 'x_flux' : conserves the flux in x-direction (eastward)
- 'y_flux' : conserves the flux in y-direction (northward)

oocgcm.core.grids.generic_2d_grid.change_grid_location_t_to_v

generic_2d_grid.**change_grid_location_t_to_v**(*scalararray*, *conserving='area'*)

Return a xarray corresponding to scalararray averaged at a new grid location.

Parameters **scalararray** : xarray.DataArray

original array to be relocated

conserving : str

any of 'area', 'x_flux' or 'y_flux'.

- 'area' : conserves the area
- 'x_flux' : conserves the flux in x-direction (eastward)
- 'y_flux' : conserves the flux in y-direction (northward)

oocgcm.core.grids.generic_2d_grid.change_grid_location_v_to_t

generic_2d_grid.**change_grid_location_v_to_t**(*scalararray*, *conserving='y_flux'*)

Return a xarray corresponding to scalararray averaged at a new grid location.

Parameters **scalararray** : xarray.DataArray

original array to be relocated

conserving : str

any of 'area', 'x_flux' or 'y_flux'.

- 'area' : conserves the area
- 'x_flux' : conserves the flux in x-direction (eastward)
- 'y_flux' : conserves the flux in y-direction (northward)

oocgcm.core.grids.generic_2d_grid.change_grid_location_f_to_u

generic_2d_grid.**change_grid_location_f_to_u**(*scalararray*, *conserving='x_flux'*)

Return a xarray corresponding to scalararray averaged at a new grid location.

Parameters **scalararray** : xarray.DataArray

original array to be relocated

conserving : str

any of 'area', 'x_flux' or 'y_flux'.

- 'area' : conserves the area
- 'x_flux' : conserves the flux in x-direction (eastward)
- 'y_flux' : conserves the flux in y-direction (northward)

oocgcm.core.grids.generic_2d_grid.change_grid_location_f_to_v

generic_2d_grid.**change_grid_location_f_to_v**(*scalararray*, *conserving='y_flux'*)

Return a xarray corresponding to scalararray averaged at a new grid location.

Parameters **scalararray** : xarray.DataArray

original array to be relocated

conserving : str

any of 'area', 'x_flux' or 'y_flux'.

- 'area' : conserves the area
- 'x_flux' : conserves the flux in x-direction (eastward)
- 'y_flux' : conserves the flux in y-direction (northward)

oocgcm.core.grids.generic_2d_grid.change_grid_location_v_to_u

generic_2d_grid.**change_grid_location_v_to_u**(*scalararray*, *conserving='area'*)

Return a xarray corresponding to scalararray averaged at a new grid location.

Parameters **scalararray** : xarray.DataArray

original array to be relocated

conserving : str

any of 'area', 'x_flux' or 'y_flux'.

- 'area' : conserves the area
- 'x_flux' : conserves the flux in x-direction (eastward)
- 'y_flux' : conserves the flux in y-direction (northward)

oocgcm.core.grids.generic_2d_grid.change_grid_location_u_to_v

generic_2d_grid.**change_grid_location_u_to_v**(*scalararray*, *conserving='area'*)

Return a xarray corresponding to scalararray averaged at a new grid location.

Parameters **scalararray** : xarray.DataArray

original array to be relocated

conserving : str

any of 'area', 'x_flux' or 'y_flux'.

- 'area' : conserves the area
- 'x_flux' : conserves the flux in x-direction (eastward)
- 'y_flux' : conserves the flux in y-direction (northward)

Vector calculus

core.grids.generic_2d_grid.norm_of_vectorfield(...)	Return the norm of a vector
core.grids.generic_2d_grid.scalar_product(...)	Return the scalar product of
core.grids.generic_2d_grid.scalar_outer_product(...)	Return the outer product of a
core.grids.generic_2d_grid.vertical_component_of_the_cross_product(...)	Return the cross product of t

oocgcm.core.grids.generic_2d_grid.norm_of_vectorfield

generic_2d_grid.**norm_of_vectorfield**(*vectorfield*)

Return the norm of a vector field, at t-point.

So far, only available for vector fields at u,v grid_location

Parameters **vectorfield** : VectorField2d namedtuple

so far only valid for vectorfield at u,v-points

oocgcm.core.grids.generic_2d_grid.scalar_product

generic_2d_grid.**scalar_product**(*vectorfield1*, *vectorfield2*)

Return the scalar product of two vector fields, at t-point.

So far, only available for vector fields at u,v grid_location.

Parameters **vectorfield1** : VectorField2d namedtuple

vectorfield2 : VectorField2d namedtuple

Notes

Multiplies each component independently, relocates each component at t grid_location then add the two products.

oocgcm.core.grids.generic_2d_grid.scalar_outer_product

generic_2d_grid.**scalar_outer_product**(*scalararray*, *vectorfield*)

Return the outer product of a scalar (t location) with a two-dimensional vector field (u,v location)

So far, only available for vector fields at u,v grid_location.

Parameters **scalararray** : xarray.DataArray

vectorfield : VectorField2d namedtuple

two-dimensional vector field at u,v grid_location

Notes

Relocates scalararray at u,v grid_location then multiply each component of vectorfield by the relocated scalararray.

oocgcm.core.grids.generic_2d_grid.vertical_component_of_the_cross_product

generic_2d_grid.**vertical_component_of_the_cross_product**(*vectorfield1*, *vectorfield2*)

Return the cross product of two vector fields.

Parameters **vectorfield1** : VectorField2d namedtuple

two-dimensional vector field at u,v grid_location

vectorfield2 : VectorField2d namedtuple

two-dimensional vector field at u,v grid_location

Notes

Relocates all the components of the VectorFields at t grid_location then compute :math:‘c = v1_x . v2_y - v1_y . v2_x.

So far, only available for vector fields at u,v grid_location.

Differential operators

core.grids.generic_2d_grid.horizontal_gradient(...)	Return the horizontal gradient of the input datas
core.grids.generic_2d_grid.horizontal_gradient_vector(...)	Return the horizontal gradient of a scalar field d
core.grids.generic_2d_grid.horizontal_gradient_tensor(...)	Return the horizontal gradient tensor of a two-di
core.grids.generic_2d_grid.horizontal_laplacian(...)	Return the horizontal laplacian of a scalar field a
core.grids.generic_2d_grid.vertical_component_of_curl(...)	Return the vertical component of the curl of a ve
core.grids.generic_2d_grid.horizontal_divergence(...)	Return the horizontal divergence of a vector fiel
core.grids.generic_2d_grid.integrate_dxdy(array)	Return the horizontal integral of array in regions
core.grids.generic_2d_grid.spatial_average_xy(array)	Return the horizontal average of array in regions

oocgcm.core.grids.generic_2d_grid.horizontal_gradient

generic_2d_grid.**horizontal_gradient**(*datastructure*)

Return the horizontal gradient of the input datastructure.

Parameters **datastructure** : xarray.DataArray or VectorField2d

Returns **result** : VectorField2d or Tensor2d

See also:

`self.horizontal_gradient_vector`, `self.horizontal_gradient_tensor`

Methods

calls horizontal_gradient_vector or horizontal_gradient_tensor depending on th type of the input datastructure.	

oocgcm.core.grids.generic_2d_grid.horizontal_gradient_vector

generic_2d_grid.**horizontal_gradient_vector**(*scalararray*)

Return the horizontal gradient of a scalar field defined at t-points.

Parameters **scalararray** : xarray.DataArray

xarray of a scalar variable at grid_location='t'

Returns **vectorfield** : VectorField2d namedtuple

x and y component of the horizontal gradient at u,v-points

oocgcm.core.grids.generic_2d_grid.horizontal_gradient_tensor

generic_2d_grid.**horizontal_gradient_tensor**(*vectorfield*)

Return the horizontal gradient tensor of a two-dimensional vector field at u,v locations.

Parameters **vectorfield** : VectorField2d

namedtuple of a vector field defined at u,v points

Returns **gradtensor** : Tensor2d

namedtuple holding the component of the horizontal gradient of the vector field at at t and f points.

Notes

The gradient of a vector field $\mathbf{u} = (u, v)$ is defined as follows :

$$\nabla \mathbf{u} = \begin{pmatrix} \partial_x u & \partial_y u \\ \partial_y v & \partial_y v \end{pmatrix}$$

where:

- $\partial_x u$ is defined at t grid location.
- $\partial_y u$ is defined at f grid location.
- $\partial_x v$ is defined at f grid location.
- $\partial_y v$ is defined at t grid location.

oocgcm.core.grids.generic_2d_grid.horizontal_laplacian

generic_2d_grid.**horizontal_laplacian**(*scalararray*)

Return the horizontal laplacian of a scalar field at t-points.

Parameters **scalararray** : xarray.DataArray

xarray of a scalar variable defined at grid_location='t'

Returns **scalararray** : xarray.DataArray

xarray of laplacian defined at grid_location='t'

Notes

Compute the laplacian as the divergence of the gradient.

oocgcm.core.grids.generic_2d_grid.vertical_component_of_curl

generic_2d_grid.**vertical_component_of_curl**(*vectorfield*)

Return the vertical component of the curl of a vector field.

Parameters **vectorfield** : VectorField2d namedtuple

Returns scalararray: xarray.DataArray

oocgcm.core.grids.generic_2d_grid.horizontal_divergence

generic_2d_grid.**horizontal_divergence**(*vectorfield*)

Return the horizontal divergence of a vector field at u,v-points.

Parameters **vectorfield** : VectorField2d namedtuple

Two-dimensional vector field at u,v-points.

Returns scalararray: xarray.DataArray

xarray of divergence at grid_location='t'

oocgcm.core.grids.generic_2d_grid.integrate_dxdy

generic_2d_grid.**integrate_dxdy**(*array*, *where=None*, *grid_location=None*, *normalize=False*)

Return the horizontal integral of array in regions where where is True.

Parameters **array** : xarray.DataArray

a dataarray with an additonal attribute specifying the grid_location. The dimension of array should include 'x' and 'y'. The shape of array should match the shape of the grid.

where: boolean xarray.DataArray

dataarray with value = True where the integration should be applied. The dimension of where should be a subset of the dimension of array. For each dimension, the size should be equal to the corresponding size of array dataarray. if where is None, the function return the integral in all the domain defined by the grid object.

grid_location : str

string describing the grid location [eg 'u','v','t','f'...]

- **if grid_location is not None** check compatibility with array.attrs.grid_location
- **if grid_location is None** use array.attrs.grid_location by default

normalize : boolean

boolean stating whether of not the integral should be normalized by the area of the region over which the integration is performed.

Returns integral: xarray.DataArray

a dataarray with reduced dimension defining the integral of array in the region of interest.

See also:

spatial_average_xy averaging over a region

oocgcm.core.grids.generic_2d_grid.spatial_average_xy

generic_2d_grid.**spatial_average_xy**(*array*, *where=None*, *grid_location=None*)

Return the horizontal average of array in regions where where is True.

Parameters **array** : xarray.DataArray

a dataarray with an additonal attribute specifying the grid_location. The dimension of array should include 'x' and 'y'. The shape of array should match the shape of the grid.

where: boolean xarray.DataArray

dataarray with value = True where the integration should be applied The dimension of where should be a subset of the dimension of array. For each dimension, the size should be equal to the corresponding size of array dataarray. if where is None, the function return the integral in all the domain defined by the grid object.

grid_location : str

string describing the grid location [eg 'u','v','t','f'...]

- **if grid_location is not None** check compatibility with array.attrs.grid_location
- **if grid_location is None** use array.attrs.grid_location by default

Returns average: xarray.DataArray

a dataarray with reduced dimension defining the average of array in the region of interest.

See also:

`integrate_dxdy` spatial integral over a region

Spatial integration

Operators specific to oceanic applications

core.grids.generic_2d_grid.geostrophic_current_from_sea_surface_height(...)	Return the geostrophic
core.grids.generic_2d_grid.q_vector_due_to_kinematic_deformation(...)	Return the component
core.grids.generic_2d_grid.frontogenesis_function(...)	Return the frontogenes

oocgcm.core.grids.generic_2d_grid.geostrophic_current_from_sea_surface_height

generic_2d_grid.**geostrophic_current_from_sea_surface_height**(*sea_surface_height*)

Return the geostrophic current on u,v-grids.

Parameters **scalararray** : xarray.DataArray

xarray of sea surface height at grid_location='t'

Returns **vectorfield** : VectorField2d namedtuple

Two-dimensional vector field of geostrophic currents at u,v-points.

oocgcm.core.grids.generic_2d_grid.q_vector_due_to_kinematic_deformation

generic_2d_grid.**q_vector_due_to_kinematic_deformation**(*velocity*, *buoyancy*)

Return the component of the generalized Q-vector associated with kinematic deformation of a two-dimensional velocity field.

Parameters **velocity** : VectorField2d

namedtuple of horisontal velocity at u,v grid locations

buoyancy : xarray.DataArray

xarray of buoyancy at grid_location='t'

Returns **vectorfield** : VectorField2d

Two-dimensional vector field of Q_{kd} at u,v-points.

Notes

Definition of the Q-vector due to kinematic deformation (following [R8], see also [R9]):

$$\mathbf{Q}^g_{kd} = -\begin{pmatrix} \partial_x u\, \partial_x b + \partial_x v_g\, \partial_y b \\ \partial_y u\, \partial_x b + \partial_y v_g\, \partial_y b \end{pmatrix}$$

References

[R8], [R9]

oocgcm.core.grids.generic_2d_grid.frontogenesis_function

generic_2d_grid.**frontogenesis_function**(*velocity*, *buoyancy*)

Return the frontogenesis function.

Parameters **velocityfield** : VectorField2d

namedtuple of horisontal velocity at u,v grid locations

buoyancy : xarray.DataArray

xarray of buoyancy at grid_location='t'

Returns **result** : xarray.DataArray

Frontogenesis function at at t location.

Notes

Definition of the frontogenesis function (following [R7])

$$F_s = \mathbf{Q}_{kd} \cdot \nabla_h b$$

References

[R7]

1.8.3 Tools for filtering timeseries and spatial fields

oocgcm.filtering.timefilters

1.8.4 Testing tools

oocgcm.test.signals

1.9 Contributing to oocgcm

If you are willing to contribute to developing oocgcm (reporting bugs, suggesting improvements, ...), here is some information.

1.9.1 Suggesting improvments to oocgcm

As for any piece of software stored on github, contributing to oocgcm involves your knowing the basics of git version control. Then just

- create a github account
- fork oocgcm repository
- push the proposed changes on your forked repository
- Submit a pull request

Contributions are also welcome on oocgcm issues tracker.

If you don't feel comfortable with the above tools, just contact me by email.

1.9.2 Overall layout of the library

Here is some information about the overall layout of oocgcm library:

```
oocgcm/
    setup.py
    oocgcm/
    docs/
    examples/
    ci/
```

`setup.py` installation script.

`oocgcm` contains the actual library

`docs` contains the documentation in rst format. Used for building the docs on readthedocs with sphinx and numpydoc

`examples` provides example of applications in notebooks and python scripts

`ci` contains the yaml files for continuous integration. Used for testing the build and testing oocgcm with several combinations of libraries.

1.9.3 Structure of `oocgcm` package

The actual package itself contains the following submodules:

```
oocgcm/
    oocgcm/
        core/
        parameters/
        griddeddata/
        oceanmodels/
        oceanfuncs/
        airseafuncs/
        spectra/
```

```
        stats/
        filtering/
        test/
```

core contains data-agnostic versions of methods that are inherently data-specific.

parameters defines physical and mathematical parameters that may be used in several submodules.

griddeddata contains data-specific methods adapted to two-dimensional gridded data, including satellite data.

oceanmodels contains data-specific methods adapted to a range of c-grid ocean models. Currently, contains only tools for NEMO ocean model.

oceanfuncs contains functions that are only relevant for analyzing ocean data but transverse to particular sources of data (a priori numpy version of the function and wrappers for xarray dataarrays).

airseafuncs will contain methods for analysing air-sea exchanges.

regrid will contain methods for regridding gridded data.

spectra will contain methods for computing wavenumber spectra and frequency spectra out of xarray.DataArray

stats will contain methods for applying descriptive statistics to gridded data.

filtering will contain methods for filtering gridded data in space or in time.

test contains the test series for oocgcm. unit tests are runs after each commit.

1.9.4 Structure of `oocgcm.core`

`oocgcm/core` contains:

```
oocgcm/core/
    io.py
    grids.py
    utils.py
```

io.py contains functions that are used for creating xarray datasets and xarray dataarrays and functions used for writing output files.

grids.py contains tools that define grid descriptor objects.

utils.py contains useful functions for several submodules. This includes function for testing and asserting types.

Files with similar names and contents can be repeated for each specific source of data when needed.

1.10 Frequently Asked Questions

1.10.1 Why is the code architecture so complex ?

The architecture and layout of oocgcm reflects the constraint that are inherent to build a tool that can be easily adapted to different sources of data.

We have decided to separate as much as possible the generic tools from the data-specific tools. Data-specific tools are generally created from generic method and try to avoid unnecessary duplication. But some method are inherently data-specific.

For instance, the 'time' dimension of NEMO ocean model output is called 'time_counter', but oocgcm assumes that the 'time' dimension is 't'. It is therefore necessary to change the name of this dimension when reading NEMO model output. This is done in oocgcm.oceanmodels.nemo.io methods.

We have chosen to gather all the methods that are specific to a certain source of data within a unique folder for facilitating the maintenance by third-parties.

In practice, switching from one model to the other should be as simple as changing

```
from oocgcm.oceanmodels.nemo import io, grids
```

into

```
from oocgcm.oceanmodels.mitgcm import io, grids
```

1.11 What's New ?

1.11.1 v0.1 (unreleased yet)

This initial release includes

- A generic two-dimensional C-grid grid descriptor that provides methods for vector calculus, differential operators and spatial integration.

CHAPTER 2

Get in touch

- Report bugs, suggest feature ideas or view the source code on GitHub.

[R8] Hoskins, B.J., Bretherton, F.P., 1972. Atmospheric Frontogenesis Models: Mathematical Formulation and Solution. J. Atmos. Sci. 29, 11-37.

[R9] Giordani, H., Prieur, L., Caniaux, G., 2006. Advanced insights into sources of vertical velocity in the ocean. Ocean Dynamics 56, 513-524.

[R7] Hoskins, B.J., Bretherton, F.P., 1972. Atmospheric Frontogenesis Models: Mathematical Formulation and Solution. J. Atmos. Sci. 29, 11-37.

Symbols

B

C

D

F

G

H