croco_xarray

March 3, 2025


```
Dante Campagnoli Napolitano
Laboratoire d'Océanographie Physique et Spatiale (LOPS)
February 2025
```


0.1 GIGATL (Gula et al., 2021)

- CROCO simulation
- Horizontal resolution, curvilinear grid of 6, 3, 1 km
- 50 (6km) and 100 (3 and 1km) terrain-following vertical levels
- SODA initial and boundary conditions
- CFSR atmospheric forcing
- k-ϵ turbulence closure scheme
- 30-s arc bathymetry (SRTM30+)
- tides, rivers, etc …

Horizontal grid

Vertical grid

[1]:
```
%load_ext memory_profiler

# processing
import numpy as np
import xarray as xr

# plotting
import matplotlib.pyplot as plt
import matplotlib.gridspec as gs

# croco support functions
import os
import sys

# append path to xtools.py
sys.path.append('../../support_func')
from xtools import *
```

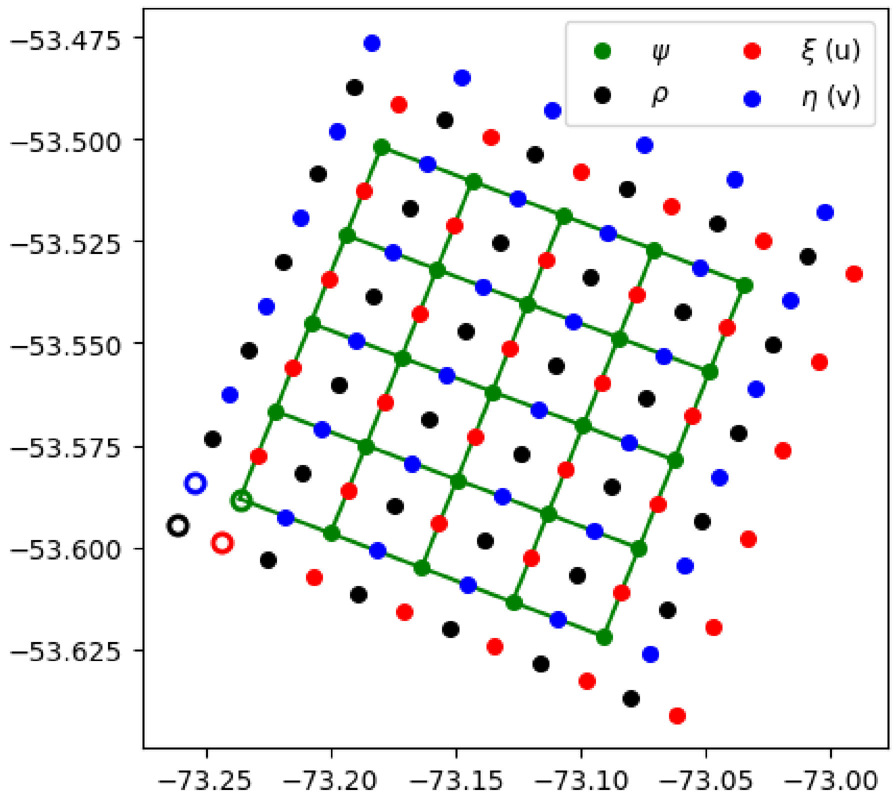
[2]:
```
# OPEN CLUSTER (Datarmor)
import dask
import dask_jobqueue
```

```
cluster = dask_jobqueue.
  ↪PBSCluster(cores=5,memory='40GB',processes=1,queue='omp', walltime='02:00:
  ↪00',interface='ib0',local_directory='$TMPDIR')
client = dask.distributed.Client(cluster)
cluster.scale(jobs=5)

client
```

[2]: <Client: 'tcp://10.148.1.89:55690' processes=0 threads=0, memory=0 B>

[3]:
```
# Initialisation GIGATL
#simulname='gigatl3' #gigatl1

# Gigatl3
path = '/home/datawork-lops-megatl/GIGATL3/'
grdfile = path+'gigatl3_grd.nc' # gigatl3_grd?
hisfile = path+'GIGATL3_1h/HIS/GIGATL3_12h_inst_2011-02-17-2011-02-21.nc'
timename='time_counter'
```

[4]:
```
# first, we look at the grid structure
pltargs={'npoints':6,'marker':'o','marker_size':30,'fig':plt.
  ↪figure(figsize=(5,5))}
visualize_grid(grdfile,pltargs)
```




```
[5]: %%html
<style>
table {float:left}
td,th {font-size: 15px}
</style>
```

```
<IPython.core.display.HTML object>
```

Variables	**x-grid** (xi_)	**y-grid** (eta_)	**z-grid** (s_)
Temperature	xi_rho	eta_rho	s_rho
u-velocity	xi_u	eta_rho	s_rho
v-velocity	xi_rho	eta_v	s_rho
ζ	xi_u	eta_v	s_rho
N^2	xi_rho	eta_rho	s_w

```
[6]: # limits CROSSROAD
limits=[-60,-40,40,52]
inds = findCROCO_index(grdfile,limits=limits)
```

```
inds: [x_min,x_max,y_min,y_max]
rho = [1332, 1902, 2965, 3512]
```

```
u = [1332, 1901, 2965, 3512]
v = [1332, 1902, 2965, 3511]
psi = [1332, 1903, 2965, 3513]
```

[7]:
```python
# choose variables to drop
drop_variables = ['time',
                  'sustr', 'svstr','bvf',
                  'lon_rho','lat_rho','lon_u','lat_u','lon_v','lat_v',
                  'time_instant','time_instant_bounds',
                  'time_counter_bounds',
                 ]

# Turn on chunking to activate dask and parallelize read/write.
chks=500
chunks = {'time':1,'xi_rho': chks,'eta_rho' :chks,'eta_v': chks, 'xi_u':␣
 ↪chks,'s_rho':1,'s_w':1}

# Slice region of interest
slicedic = dict(xi_rho=slice(inds['rho'][0],inds['rho'][1]),␣
 ↪eta_rho=slice(inds['rho'][2],inds['rho'][3]),
                xi_u=slice(inds['u'][0],inds['u'][1]),␣
 ↪eta_v=slice(inds['v'][2],inds['v'][3]),)

ds = xr.open_dataset(hisfile,drop_variables=drop_variables)
ds = adjust_coords(ds)
ds = ds.isel(slicedic).chunk(chunks)
```

for XIOS files

[8]:
```python
ds
```

[8]:
```
<xarray.Dataset>
Dimensions:  (eta_rho: 547, xi_rho: 570, xi_u: 569, eta_v: 546, s_rho: 100,
              s_w: 101, time: 10)
Coordinates:
    lat_rho  (eta_rho, xi_rho) float32 dask.array<chunksize=(500, 500),
meta=np.ndarray>
    lon_rho  (eta_rho, xi_rho) float32 dask.array<chunksize=(500, 500),
meta=np.ndarray>
    lat_u    (eta_rho, xi_u) float32 dask.array<chunksize=(500, 500),
meta=np.ndarray>
    lon_u    (eta_rho, xi_u) float32 dask.array<chunksize=(500, 500),
meta=np.ndarray>
    lat_v    (eta_v, xi_rho) float32 dask.array<chunksize=(500, 500),
meta=np.ndarray>
```

```
    lon_v    (eta_v, xi_rho) float32 dask.array<chunksize=(500, 500),
meta=np.ndarray>
  * s_rho    (s_rho) float32 -0.995 -0.985 -0.975 … -0.025 -0.015 -0.005
    lat_w    (eta_rho, xi_rho) float32 dask.array<chunksize=(500, 500),
meta=np.ndarray>
    lon_w    (eta_rho, xi_rho) float32 dask.array<chunksize=(500, 500),
meta=np.ndarray>
  * s_w      (s_w) float32 -1.0 -0.99 -0.98 -0.97 … -0.03 -0.02 -0.01 0.0
  * time     (time) datetime64[ns] 2011-02-17T12:00:00 2011-02-18 … 2011-02-22
Dimensions without coordinates: eta_rho, xi_rho, xi_u, eta_v
Data variables:
    zeta     (time, eta_rho, xi_rho) float32 dask.array<chunksize=(1, 500, 500),
meta=np.ndarray>
    ubar     (time, eta_rho, xi_u) float32 dask.array<chunksize=(1, 500, 500),
meta=np.ndarray>
    vbar     (time, eta_v, xi_rho) float32 dask.array<chunksize=(1, 500, 500),
meta=np.ndarray>
    u        (time, s_rho, eta_rho, xi_u) float32 dask.array<chunksize=(1, 1,
500, 500), meta=np.ndarray>
    v        (time, s_rho, eta_v, xi_rho) float32 dask.array<chunksize=(1, 1,
500, 500), meta=np.ndarray>
    w        (time, s_rho, eta_rho, xi_rho) float32 dask.array<chunksize=(1, 1,
500, 500), meta=np.ndarray>
    temp     (time, s_rho, eta_rho, xi_rho) float32 dask.array<chunksize=(1, 1,
500, 500), meta=np.ndarray>
    salt     (time, s_rho, eta_rho, xi_rho) float32 dask.array<chunksize=(1, 1,
500, 500), meta=np.ndarray>
    rho      (time, s_rho, eta_rho, xi_rho) float32 dask.array<chunksize=(1, 1,
500, 500), meta=np.ndarray>
Attributes: (12/45)
    name:            ./HIS/GIGATL3_12h_inst
    description:     Created by xios
    Conventions:     CF-1.6
    timeStamp:       2020-Mar-11 18:50:21 GMT
    uuid:            934d8b1e-e339-40b7-83e4-1e1ad472002d
    title:           GIGATL3
    …               …
    SRCS:            main.F step.F read_inp.F timers_roms.F init_scalars.F ini…
    CPP-options:     REGIONAL GIGATL3 MPI_TIME MPI XIOS XIOS2 OBC_EAST OBC_NOR…
    sc_w:            [-1.   -0.99 -0.98 -0.97 -0.96 -0.95 -0.94 -0.93 -0.92 -0…
    Cs_w:            [-1.00000000e+00 -9.83735238e-01 -9.66697847e-01 -9.48934…
    sc_r:            [-0.995 -0.985 -0.975 -0.965 -0.955 -0.945 -0.935 -0.925 …
    Cs_r:            [-9.91966929e-01 -9.75310303e-01 -9.57903911e-01 -9.39797…
```

- for this tutorial, we will choose one snapshot

[9]:
```
ds = ds.isel(time=0)
```

[10]:
```
# Read gridfile
grd = adjust_coords(xr.open_dataset(grdfile))
grd = grd.rename({'xi_v':'xi_rho','eta_u':'eta_rho'}) # check if adjust_coords␣
 ↪is doing the job
grd = grd.isel(slicedic)

# another way of cutting a region
# maskrho=grd.mask_rho.where( (grd.eta_rho>=inds['rho'][2]) & (grd.
 ↪eta_rho<inds['rho'][3]) &
#                              (grd.xi_rho >=inds['rho'][0]) & (grd.xi_rho␣
 ↪<inds['rho'][1]), drop=True)

maskrho = grd.mask_rho.compute().astype('bool')

# shift mask for u and v grids
masku = maskrho.isel(xi_rho = slice(None,-1)).rename(xi_rho='xi_u').
 ↪astype('bool')
maskv = maskrho.isel(eta_rho = slice(None,-1)).rename(eta_rho='eta_v').
 ↪astype('bool')

# depending on the dataset and masks, we have to apply it later directly on the␣
 ↪variable
# ds = ds.where(maskrho)
```

for regular CROCO files

[11]:
```
# if 'h' not in ds:
#     grd = adjust_coords(xr.open_dataset(grdfile))
#     ds = add_grd(ds,grd.isel(slicedic))

# add vertical grid to dataset and calculate z
ds = add_grd(ds,grd)

# ---> if you have more than one timestep, you can choose ds.isel(time=0), it's␣
 ↪less precise (because z varies slightly with time, but it's way faster to␣
 ↪compute)
zrho,zw = calc_vertical_coord(ds)
```

[12]:
```
%%time

# may be optimized by playing with chunks
zrho = zrho.compute()
zw = zw.compute()
```

```
CPU times: user 1.85 s, sys: 2.11 s, total: 3.96 s
```

```
Wall time: 20.9 s
```

Calculate vertical velocity
xtools : **get_w**

```
[13]: # add vertical velocity to dataset
ds['w'] = getw(ds.u,ds.v,ds.pm,ds.pn,zrho,zw)
```

Calculate potential density
xtools : **rho_eos**

note: function not adapted to other reference values σ_1, σ_2 … nor neutral density …

```
[14]: import gsw
```

```
[18]: SA=gsw.SA_from_SP(ds.salt,zrho,ds.lon_rho,ds.lat_rho)
```

```
[20]: sig1=gsw.pot_rho_t_exact(SA,ds.temp,zrho,1000)
```

```
[23]: sig1=sig1.chunk({'xi_rho':sig1.xi_rho.size,'eta_rho':sig1.eta_rho.size})
sig1
```

```
[23]: <xarray.DataArray 'salt' (s_rho: 100, eta_rho: 547, xi_rho: 570)>
dask.array<rechunk-merge, shape=(100, 547, 570), dtype=float64, chunksize=(1,
547, 570), chunktype=numpy.ndarray>
Coordinates:
  * s_rho    (s_rho) float32 -0.995 -0.985 -0.975 … -0.025 -0.015 -0.005
    lat_rho  (eta_rho, xi_rho) float32 dask.array<chunksize=(547, 570),
meta=np.ndarray>
    lon_rho  (eta_rho, xi_rho) float32 dask.array<chunksize=(547, 570),
meta=np.ndarray>
    lat_w    (eta_rho, xi_rho) float32 dask.array<chunksize=(547, 570),
meta=np.ndarray>
    lon_w    (eta_rho, xi_rho) float32 dask.array<chunksize=(547, 570),
meta=np.ndarray>
    time     datetime64[ns] 2011-02-17T12:00:00
Dimensions without coordinates: eta_rho, xi_rho
Attributes:
    long_name:           salinity
    units:               PSU
    online_operation:    instant
    interval_operation:  12 h
    interval_write:      12 h
    cell_methods:        time: point
```

```
[24]: # add sigma0 density (usage example)
# ds['rho'] = rho_eos(ds.temp,ds.salt,zrho,9.81,ds.
 ↪attrs['rho0'],z_w=None,sig0=True)

# ---> but this output already has the rho as a density anomaly, we just  need␣
 ↪to fix it by adding the rho0 constant ...
ds['rho'] = ds.rho+ds.attrs['rho0']
```

```
[25]: %%time

# one depth (around isopycnal sig1 = 32.43 in Solodoch et al 2020, Fig 5)
znew = xr.DataArray([-1450],dims='s_rho')
temp_z = interpolate_vertically(ds.
 ↪temp,zrho,znew,ztype='depth',interp_boundaries=False)

# rechunk
temp_z = temp_z.chunk({'eta_rho':temp_z.eta_rho.size,'xi_rho':temp_z.xi_rho.
 ↪size})
```

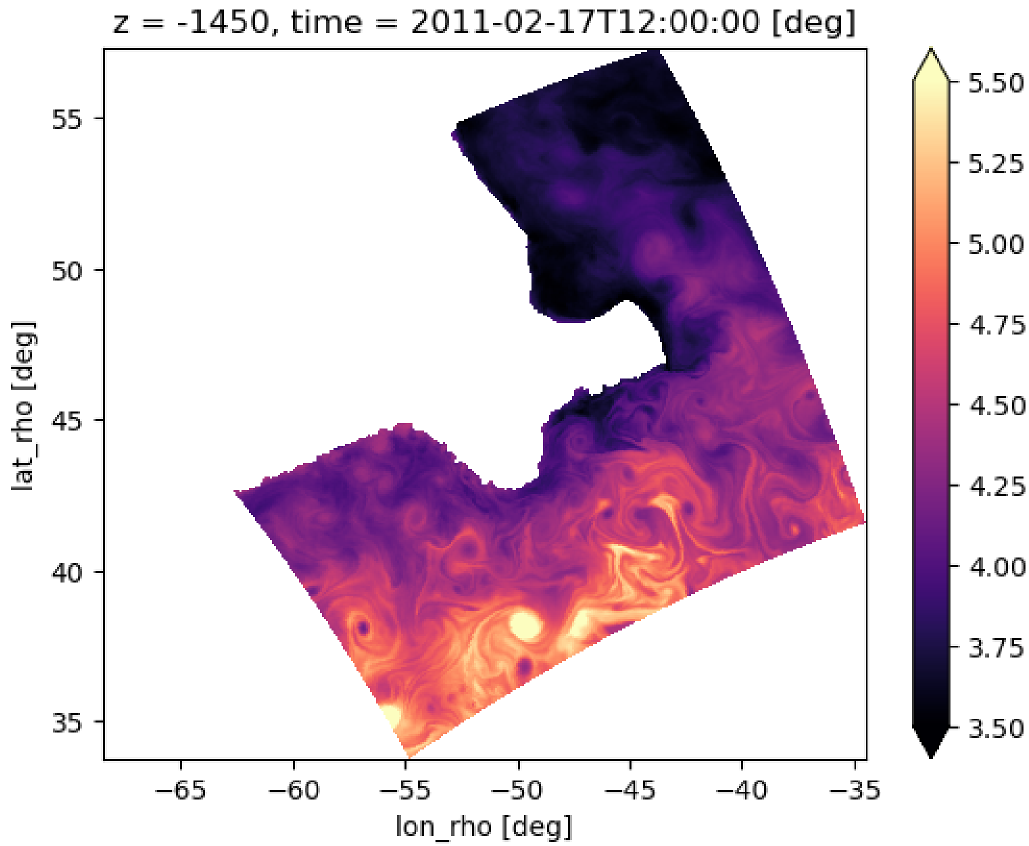
```
CPU times: user 1.82 s, sys: 1.03 s, total: 2.85 s
Wall time: 2.74 s
```

```
[26]: temp_z
```

```
[26]: <xarray.DataArray (z: 1, eta_rho: 547, xi_rho: 570)>
dask.array<rechunk-merge, shape=(1, 547, 570), dtype=float64, chunksize=(1, 547,
570), chunktype=numpy.ndarray>
Coordinates:
  * z        (z) int64 -1450
Dimensions without coordinates: eta_rho, xi_rho
```

```
[27]: # beware with interpolated values under topography. We can create a mask for␣
 ↪the desired level (or use interp_boundaries=False when interpolating)
# maskz = xr.where(grd.h>1450,True,False)

# add lon-lat coordinates
temp_z  = temp_z.assign_coords(lon_rho=ds.lon_rho,lat_rho=ds.lat_rho)

# since we did not compute(), it takes some time to plot ...

# temp_z.where(maskz).squeeze().plot.pcolormesh(vmin=3.5,vmax=5.
 ↪5,cmap='magma',x='lon_rho',y='lat_rho')
temp_z.squeeze().plot.pcolormesh(x='lon_rho',y='lat_rho',vmin=3.5,vmax=5.
 ↪5,cmap='magma')
```

```
[27]: <matplotlib.collections.QuadMesh at 0x2aab4f723c40>
```

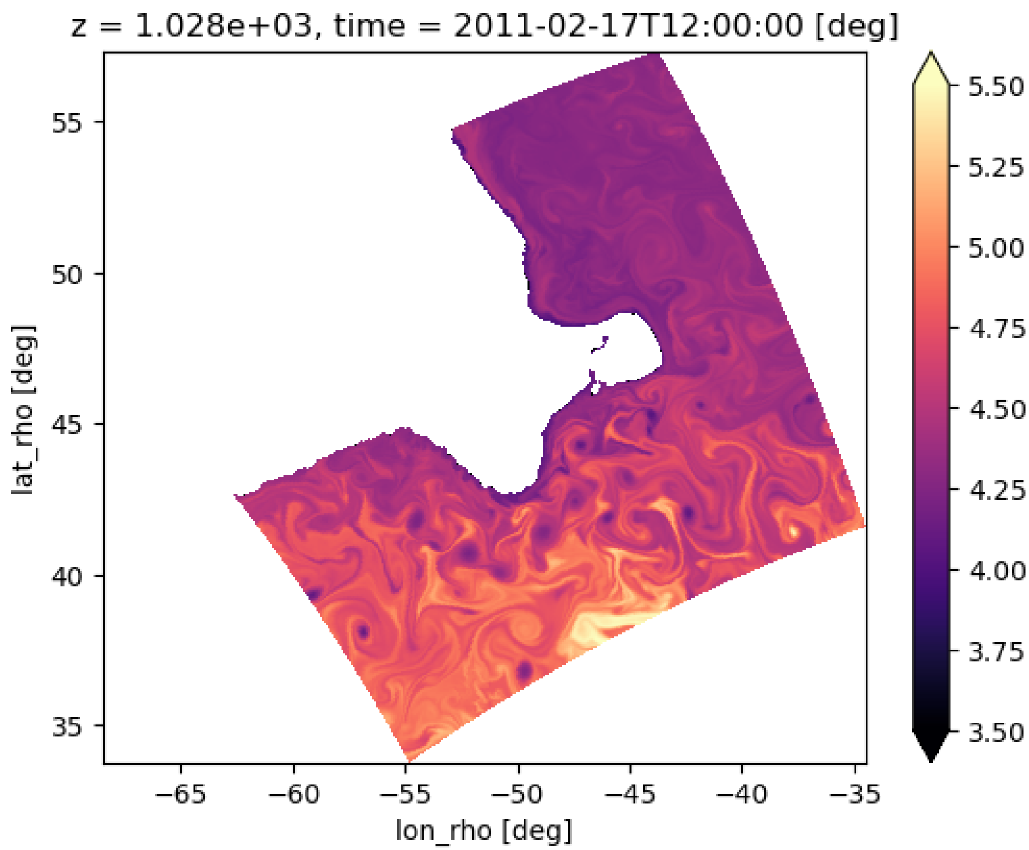


[28]:
```
%%time

# one isopycnal (27.68, upper boundary of LSW, Solodoch et al 2020, Sec 3b )
iso=xr.DataArray([1027.68],dims='s_rho')
temp_iso = interpolate_vertically(ds.temp,ds.
  ↪rho,iso,ztype='density',interp_boundaries=False)

# rechunk
temp_iso = temp_iso.chunk({'eta_rho':temp_iso.eta_rho.size,'xi_rho':temp_iso.
  ↪xi_rho.size})
```

```
CPU times: user 64 ms, sys: 0 ns, total: 64 ms
Wall time: 63.7 ms
```

[29]:
```
# add lon-lat coordinates
temp_iso  = temp_iso.assign_coords(lon_rho=ds.lon_rho,lat_rho=ds.lat_rho)

temp_iso.squeeze().plot.pcolormesh(x='lon_rho',y='lat_rho',vmin=3.5,vmax=5.
  ↪5,cmap='magma')
```

[29]: <matplotlib.collections.QuadMesh at 0x2aab64703a30>



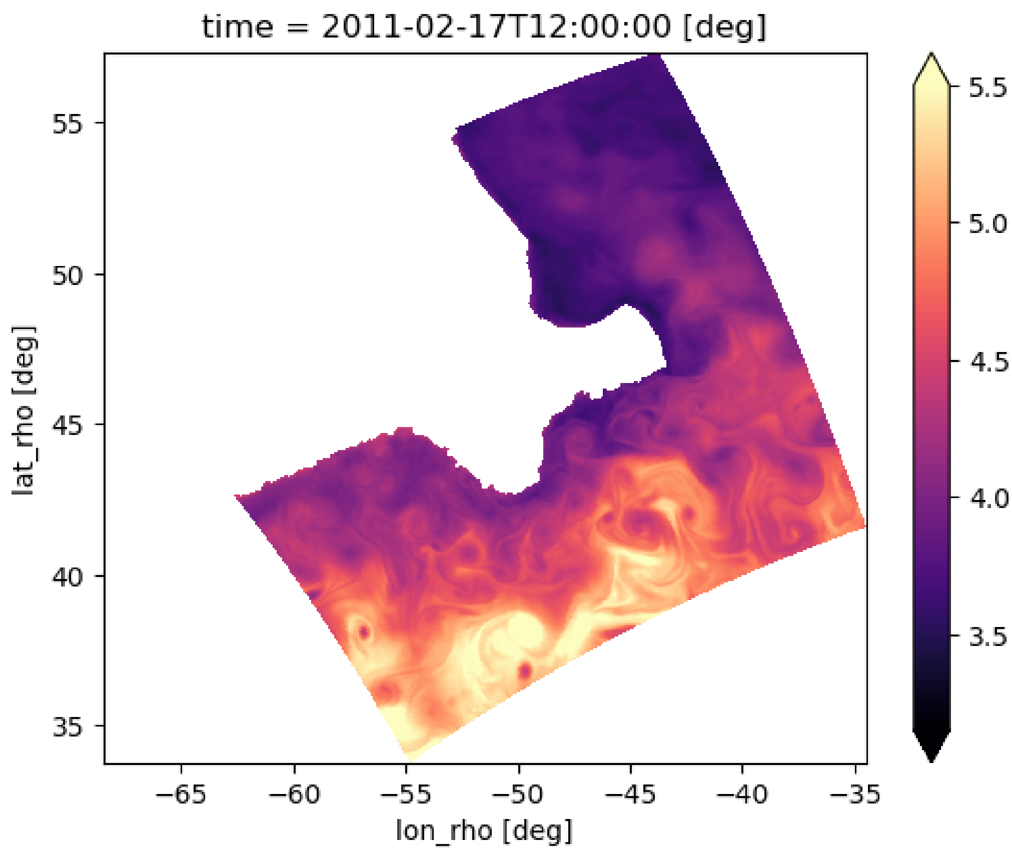
[30]:
```
%%time

# average to get mean depth (Fig 5a Solodoch et al)
zlims = np.array([-800,-2100])
temp_zavg = layer_average(ds.temp,zrho,zw,zlims,ztype='depth')

# beware with interpolated values under topography. We can create a mask for
  ↪the desired mean level
maskz = xr.where(grd.h>1450,True,False) # or interp depth with option
  ↪interp_boundaries=False
```

```
CPU times: user 2.68 s, sys: 2.09 s, total: 4.77 s
Wall time: 3.77 s
```

[31]:
```
# add mask and lon-lat coordinates
temp_zavg  = temp_zavg.where(maskz).assign_coords(lon_rho=ds.lon_rho,lat_rho=ds.
  ↪lat_rho)
```

```
temp_zavg.squeeze().plot.pcolormesh(x='lon_rho',y='lat_rho',vmax=3.5,vmin=5.
↪5,cmap='magma')
```

[31]: <matplotlib.collections.QuadMesh at 0x2aab64bdab00>



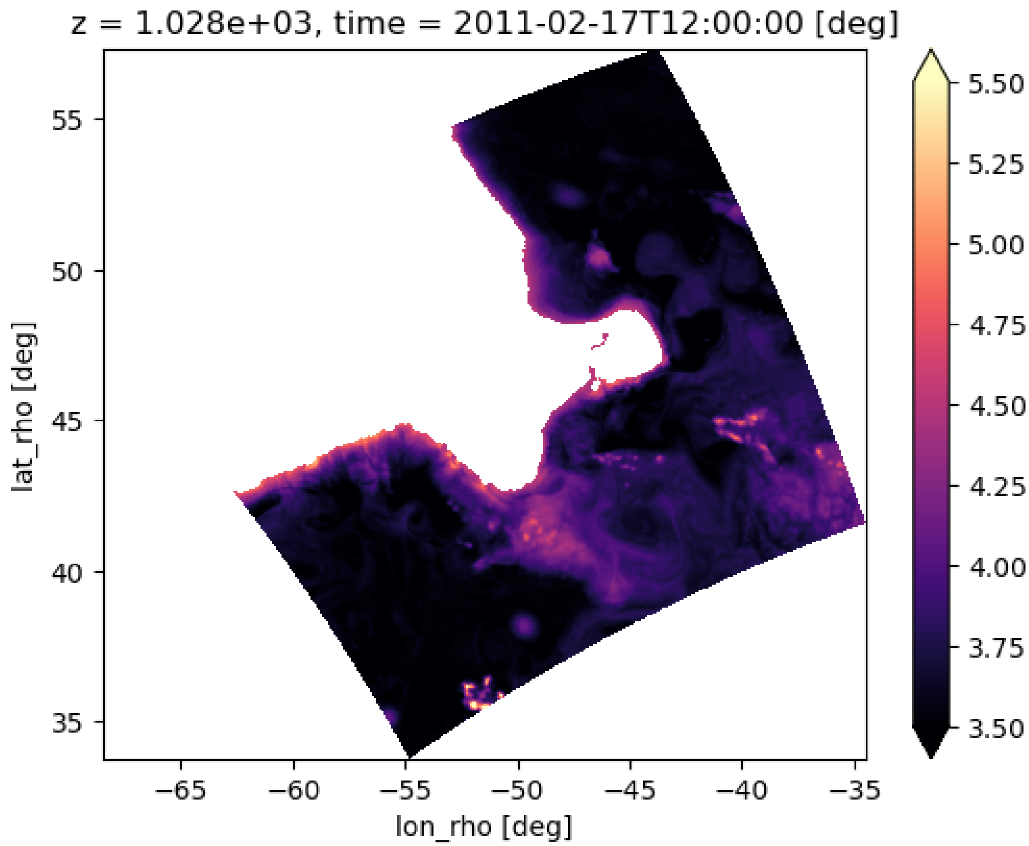
[32]:
```
%%time
# mean isopycnal
isop=[1027.0,1028.36]
temp_iavg = layer_average(ds.temp,zrho,zw,dens=ds.
↪rho,zlims=isop,ztype='density')
```

```
CPU times: user 13.8 s, sys: 1.01 s, total: 14.8 s
Wall time: 20 s
```

[33]:
```
iso = [1027.68]
mask_iso = interpolate_vertically(ds.rho,ds.
↪rho,iso,ztype='density',interp_boundaries=False)
mask_iso = np.isnan(mask_iso).squeeze()
```

```
[34]: # add mask and lon-lat coordinates
temp_iavg  = temp_iavg.where(~mask_iso).assign_coords(lon_rho=ds.
 ↪lon_rho,lat_rho=ds.lat_rho)

temp_iavg.squeeze().plot.pcolormesh(x='lon_rho',y='lat_rho',vmin=3.5,vmax=5.
 ↪5,cmap='magma')
```

```
[34]: <matplotlib.collections.QuadMesh at 0x2aab69671e70>
```



note: ***get_vertical_section*** **requires the installation of the** ***xoak*** **package**

```
[35]: # First, we choose two transects to interpolate

# model average dx
dx = grd.lon_rho.diff('xi_rho').mean().values#*111000*np.cos(np.deg2rad(47)) ~␣
 ↪3km

plon,plat=[-46.,-40.],[47.5,47.5]
lon_rad1 = np.arange(plon[0],plon[-1]+dx,dx)
lat_rad1 = plat[0]*np.ones_like(lon_rad1)
```

```
plon,plat=[-50.5,-50.5],[38.,44.]
lat_rad2 = np.arange(plat[0],plat[-1]+dx,dx)
lon_rad2 = plon[0]*np.ones_like(lat_rad2)
```

[36]:
```
%%time

# Then we extract the vertical sections (in sigma levels)

# get temperature vertical section
trad1 = get_vertical_section(ds.temp,lon_rad1,lat_rad1)
trad2 = get_vertical_section(ds.temp,lon_rad2,lat_rad2)

# so we need the corresponding depth of each sigma level to plot ...

# get depth vertical section
zrad1 = get_vertical_section(zrho,lon_rad1,lat_rad1)
zrad2 = get_vertical_section(zrho,lon_rad2,lat_rad2)

# make distance section 2D to plot (not sure if necessary with pcolormesh)
DD1 = xr.ones_like(zrad1)*trad1.dist
DD2 = xr.ones_like(zrad2)*trad1.dist
```

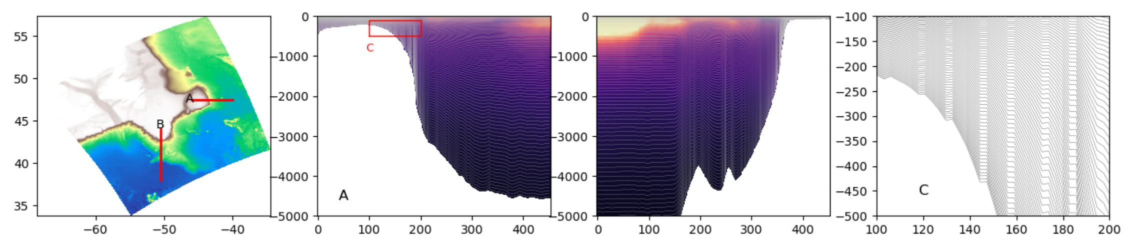
```
CPU times: user 6.52 s, sys: 1.96 s, total: 8.48 s
Wall time: 1min 18s
```

[37]:
```
fig,ax = plt.subplots(1,4,figsize=(16,3))

ax[0].pcolormesh(grd.lon_rho,grd.lat_rho,grd.h,cmap='terrain_r')
ax[0].plot(lon_rad1,lat_rad1,lw=2,c='r'); ax[0].annotate('A',xy=(-46.2,47.
 ↪6,),xycoords='data',ha='center',va='center')
ax[0].plot(lon_rad2,lat_rad2,lw=2,c='r'); ax[0].annotate('B',xy=(-50.6,44.
 ↪5,),xycoords='data',ha='center',va='center')

ax[1].pcolormesh(DD1,zrad1,trad1.T,cmap='magma',vmin=1,vmax=16);
ax[1].
 ↪plot([100,100,200,200,100],[-500,-100,-100,-500,-500],c='r',lw=1,zorder=1e3)
ax[1].plot(DD1,zrad1,lw=0.2,c='0.75');
ax[1].set_ylim(-5000,0)
ax[1].
 ↪annotate('A',xy=(50,-4500),xycoords='data',ha='center',va='center',fontsize=12)
ax[1].
 ↪annotate('C',xy=(100,-800),xycoords='data',ha='center',va='center',fontsize=9,color='red')

ax[2].pcolormesh(DD2,zrad2,trad2.T,cmap='magma',vmin=1,vmax=16);
ax[2].plot(DD2,zrad2,lw=0.2,c='0.75');
```

```
ax[2].
 ↪annotate('B',xy=(617,-4500),xycoords='data',ha='center',va='center',fontsize=12)
ax[2].set_ylim(-5000,0)

ax[3].plot(DD1,zrad1,lw=0.2,c='k');
ax[3].set_xlim(100,200)
ax[3].set_ylim(-500,-100)
ax[3].
 ↪annotate('C',xy=(120,-450),xycoords='data',ha='center',va='center',fontsize=12)
```

```
/dev/shm/pbs.2350798.datarmor0/ipykernel_12072/516839467.py:7: UserWarning: The
input coordinates to pcolormesh are interpreted as cell centers, but are not
monotonically increasing or decreasing. This may lead to incorrectly calculated
cell edges, in which case, please supply explicit cell edges to pcolormesh.
  ax[1].pcolormesh(DD1,zrad1,trad1.T,cmap='magma',vmin=1,vmax=16);
/dev/shm/pbs.2350798.datarmor0/ipykernel_12072/516839467.py:14: UserWarning: The
input coordinates to pcolormesh are interpreted as cell centers, but are not
monotonically increasing or decreasing. This may lead to incorrectly calculated
cell edges, in which case, please supply explicit cell edges to pcolormesh.
  ax[2].pcolormesh(DD2,zrad2,trad2.T,cmap='magma',vmin=1,vmax=16);
```

[37]: Text(120, -450, 'C')



[38]:
```
#### cases

# Reinterpolate temperature (rho grid) to either u v, or psi grids

# temp_u   = rho2u('temp', ds)
# temp_v   = rho2v('temp', ds)
# temp_psi = rho2psi('temp', ds)

# Reinterpolate velocity (u grid, v grid) or vorticity (psi grid) to rho grid

# u_rho    = u2rho('u', ds)
# v_rho    = v2rho('v', ds)
# zeta_rho = psi2rho(zeta, ds= N<p align="center">one)
```

[39]:
```
ds.u
```

[39]:
```
<xarray.DataArray 'u' (s_rho: 100, eta_rho: 547, xi_u: 569)>
dask.array<getitem, shape=(100, 547, 569), dtype=float32, chunksize=(1, 500,
500), chunktype=numpy.ndarray>
Coordinates:
  * s_rho    (s_rho) float32 -0.995 -0.985 -0.975 … -0.025 -0.015 -0.005
    lat_u    (eta_rho, xi_u) float32 dask.array<chunksize=(500, 500),
meta=np.ndarray>
    lon_u    (eta_rho, xi_u) float32 dask.array<chunksize=(500, 500),
meta=np.ndarray>
    time     datetime64[ns] 2011-02-17T12:00:00
Dimensions without coordinates: eta_rho, xi_u
Attributes:
    long_name:           u-momentum component
    units:               meter second-1
    online_operation:    instant
    interval_operation:  12 h
    interval_write:      12 h
    cell_methods:        time: point
```

[40]:
```
u_rho = u2rho('u', ds)
u_rho
```

[40]:
```
<xarray.DataArray 'u' (s_rho: 100, eta_rho: 547, xi_rho: 570)>
dask.array<concatenate, shape=(100, 547, 570), dtype=float32, chunksize=(1, 500,
500), chunktype=numpy.ndarray>
Dimensions without coordinates: s_rho, eta_rho, xi_rho
```

[41]:
```
znew = xr.DataArray([-1500,-1400],dims='s_rho')
temp_z = interpolate_vertically(ds.
 ↪temp,zrho,znew,ztype='depth',interp_boundaries=False)

temp_z = temp_z.chunk({'eta_rho':temp_z.eta_rho.size,'xi_rho':temp_z.xi_rho.
 ↪size}).rename({'z':'s_rho'})

# beware with interpolated values under topography. We can create a mask for␣
 ↪the desired mean level
maskz = xr.where(grd.h>1500,True,False) # or interp depth with option␣
 ↪interp_boundaries=False
```

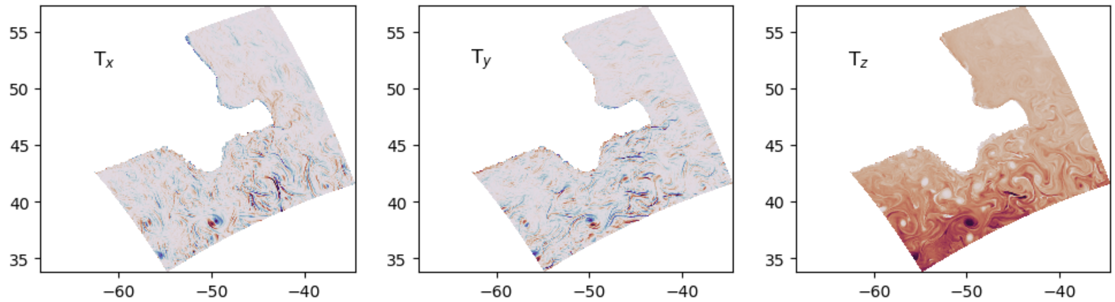
[42]:
```
%%time

### Zonal derivative dx
Tx = diffx(temp_z,ds.pm)

Tx = Tx.chunk({'eta_rho':Tx.eta_rho.size,'xi_u':Tx.xi_u.size})
```

```
Tx = Tx.compute()

### Meridional derivative dy
Ty = diffy(temp_z,ds.pn)

Ty = Ty.chunk({'eta_v':Ty.eta_v.size,'xi_rho':Ty.xi_rho.size})
Ty = Ty.compute()

### Vertical derivative dz
Tz = diffz(temp_z,znew)

Tz = Tz.chunk({'eta_rho':Tz.eta_rho.size,'xi_rho':Tz.xi_rho.size})
Tz = Tz.compute()
```

```
CPU times: user 17 s, sys: 1.54 s, total: 18.5 s
Wall time: 37.9 s
```

[43]:
```
#plt.get_cmap('twilight_shifted')

fig,ax = plt.subplots(1,3,figsize=(12,3))

ax[0].pcolormesh(grd.lon_u,grd.lat_u,Tx.
 ↪isel(s_rho=0),vmin=-5e-5,vmax=5e-5,cmap='twilight_shifted')
ax[0].annotate('T$_x$',xy=(0.2,0.8),xycoords='axes␣
 ↪fraction',ha='center',va='center',fontsize=12)

ax[1].pcolormesh(grd.lon_v,grd.lat_v,Ty.
 ↪isel(s_rho=0),vmin=-5e-5,vmax=5e-5,cmap='twilight_shifted')
ax[1].annotate('T$_y$',xy=(0.2,0.8),xycoords='axes␣
 ↪fraction',ha='center',va='center',fontsize=12)

pc=ax[2].pcolormesh(grd.lon_rho,grd.lat_rho,Tz.
 ↪squeeze(),vmin=-5e-3,vmax=5e-3,cmap='twilight_shifted')
ax[2].annotate('T$_z$',xy=(0.2,0.8),xycoords='axes␣
 ↪fraction',ha='center',va='center',fontsize=12)

# plt.colorbar(pc,ax=ax[2])
```

[43]:
```
Text(0.2, 0.8, 'T$_z$')
```



Derivatives on sigma-levels
xtools : **diffxi_sig** , **diffeta_sig**

---

note: diffz is the same for (x,y) interpolated on a z-grid or on the original grid

[35]:
```
# usage example adding the "zrho" matrix (not computing because will take too␣
↪long)

Txi = diffxi_sig(ds.temp,ds.pm,zrho)
Teta = diffeta_sig(ds.temp,ds.pn,zrho)
```

---

1 Other utility ...

Rotate velocities (to plot on a cartesian grid)

- xtools : **rotuv**

Calculate distance

- xtools : **calc_dist**

Steal Jonathan's netcdf colormaps

- xtools: **nc_colormap** (need to have folder with colormaps alongside xtools.py)

1.1 ## Practical examples using some xtools

1) Calculate horizontal Temperature advection along isopycnal

$$\mathbf{u} \cdot \nabla T = u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y}$$

---

[44]:
```
# interpolate u, v and T on isopycnal (keep original grid for now)
```

```
temp_iso = interpolate_vertically(ds.temp,ds.
 ↪rho,iso,ztype='density',interp_boundaries=False)

u_iso    = interpolate_vertically(ds.u,rho2u(ds.
 ↪rho),iso,ztype='density',interp_boundaries=False)
v_iso    = interpolate_vertically(ds.v,rho2v(ds.
 ↪rho),iso,ztype='density',interp_boundaries=False)

# temp_iso = temp_iso.chunk({'eta_rho':temp_iso.eta_rho.size,'xi_rho':temp_iso.
 ↪xi_rho.size})
# u_iso = u_iso.chunk({'eta_rho':u_iso.eta_rho.size,'xi_u':u_iso.xi_u.size})
# v_iso = v_iso.chunk({'eta_v':v_iso.eta_v.size,'xi_rho':v_iso.xi_rho.size})

# x and y gradient
Tx = diffx(temp_iso,ds.pm)
Ty = diffy(temp_iso,ds.pn)

# change grid to sum Tx and Ty
T_adv = u2rho(u_iso*Tx) + v2rho(v_iso*Ty)

# resize chunks for better computation
T_adv = T_adv.chunk({'eta_rho':T_adv.eta_rho.size,'xi_rho':T_adv.xi_rho.size})
```

[45]:
```
%%time

T_adv = T_adv.compute()
```

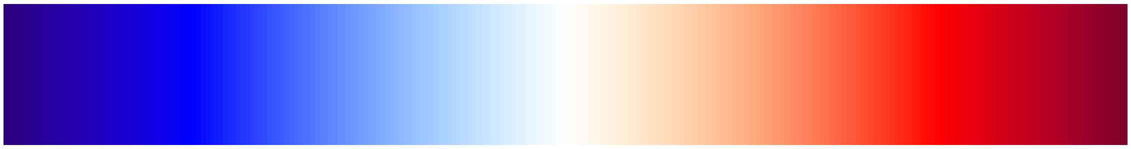
```
CPU times: user 17.6 s, sys: 224 ms, total: 17.8 s
Wall time: 50.6 s
```

[46]:
```
# mask to plot
iso = [1027.68]
mask_iso = interpolate_vertically(ds.rho,ds.
 ↪rho,iso,ztype='density',interp_boundaries=False)
mask_iso = np.isnan(mask_iso).squeeze()
```

Plot T_adv

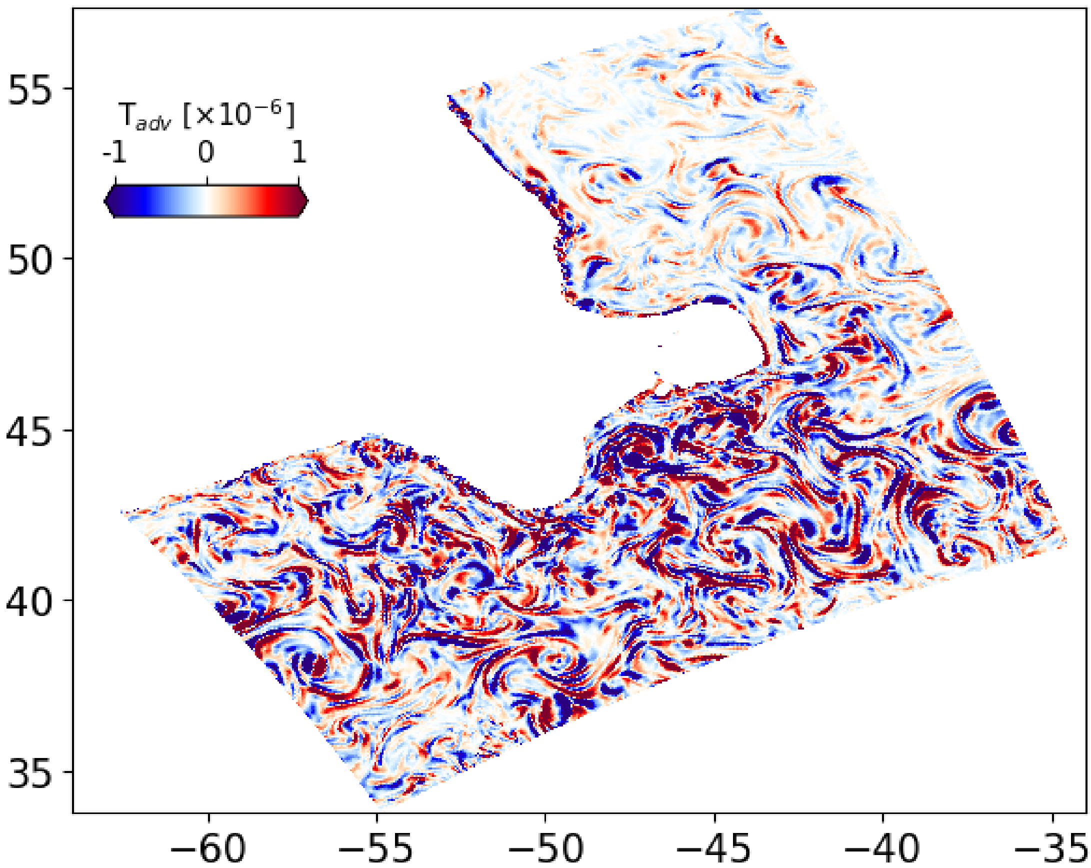
[47]:
```
font={'weight':'normal','size':15}
plt.rc('font',**font)

cmap_grad=nc_colormap('blu_red')
cmap_grad
```

[47]:



[48]:

```
fig = plt.figure(figsize=(7,0.8*7),facecolor='w')
gsp = gs.GridSpec(nrows=51,ncols=41,figure=fig)

ax = fig.add_subplot(gsp[:,:])

pc=ax.pcolormesh(grd.lon_rho,grd.lat_rho,1e6*T_adv.
 ↪squeeze(),vmin=-1,vmax=1,cmap=cmap_grad)
ax.set_xlim([-64,-34])

cax = fig.add_axes([ax.bbox.x0/fig.bbox.width+0.025,ax.bbox.y1/fig.bbox.
 ↪height-0.2,
                    0.2*ax.bbox.width/fig.bbox.width,0.03])

cbar = plt.colorbar(pc,cax=cax,orientation='horizontal',extend='both')
cbar.set_ticks([-1,0,1]);cbar.set_ticklabels(['-1','0','1'],fontsize=11)
cbar.ax.xaxis.set_ticks_position('top')
cbar.set_label(r'T$_{adv}$ [$\times 10^{-6}$]',fontsize=11,labelpad=5)
cbar.ax.xaxis.set_label_position('top')
```



[ ]:
```
# interpolate u, v and T on isopycnal (keep original grid for now)

# (calculated a few cells above)
# temp_iso = interpolate_vertically(ds.temp,ds.
 ↪rho,iso,ztype='density',interp_boundaries=False)
# u_iso    = interpolate_vertically(ds.u,rho2u(ds.
 ↪rho),iso,ztype='density',interp_boundaries=False)
# v_iso    = interpolate_vertically(ds.v,rho2v(ds.
 ↪rho),iso,ztype='density',interp_boundaries=False)
```

[49]:
```
# f0 [2D]
f0 = 2*7.29e-5*np.sin(np.deg2rad(grd.lat_rho))

# differentiate
dudx = diffx(u_iso.squeeze(),ds.pm,grid='u',coords=False) # dudx is in rho-grid
dvdx = diffx(v_iso.squeeze(),ds.pm,grid='v',coords=False) # dvdx is in psi-grid
dudy = diffy(u_iso.squeeze(),ds.pn,grid='u',coords=False) # dudy is in psi-grid
dvdy = diffy(v_iso.squeeze(),ds.pn,grid='v',coords=False) # dvdy is in rho-grid
```

```
[51]: ### 2.1) Vorticity (originally in psi-grid, but we will change to rho for␣
       ↪plotting)

      Vort = psi2rho(dvdx - dudy) / f0

      # to have zeta in the same size and div and alpha
      Vort = Vort.isel(xi_rho=slice(1,-1),eta_rho=slice(1,-1)).T

      Vort = Vort.chunk({'eta_rho':Vort.eta_rho.size,'xi_rho':Vort.xi_rho.size})
```

```
[52]: ### 2.2) Divergence

      Dive = (dudx.isel(eta_rho=slice(1,-1)) + dvdy.isel(xi_rho=slice(1,-1)))/ f0.
       ↪isel(xi_rho=slice(1,-1),eta_rho=slice(1,-1))

      Dive = Dive.chunk({'eta_rho':Dive.eta_rho.size,'xi_rho':Dive.xi_rho.size})
```

```
[53]: ### 2.3) Strain

      Stra = np.sqrt((dudx.isel(eta_rho=slice(1,-1)) - dvdy.
       ↪isel(xi_rho=slice(1,-1)))**2 +
                     (psi2rho(dvdx + dudy).
       ↪isel(xi_rho=slice(1,-1),eta_rho=slice(1,-1)))**2
                    )/ f0.isel(xi_rho=slice(1,-1),eta_rho=slice(1,-1))

      Stra = Stra.chunk({'eta_rho':Stra.eta_rho.size,'xi_rho':Stra.xi_rho.size})
```

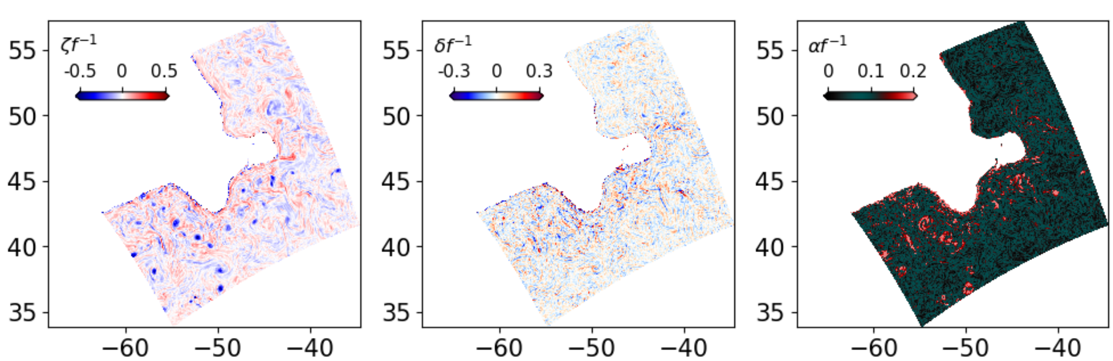
```
[54]: %%time

      # Compute calculations
      [Vort,Dive,Stra] = map(lambda da: da.compute(), [Vort,Dive,Stra])
```

```
CPU times: user 44.1 s, sys: 1.03 s, total: 45.1 s
Wall time: 1min 57s
```

```
[55]: cmap_vrt=nc_colormap('blue_red')
      cmap_div=nc_colormap('blu_red')
      cmap_str=nc_colormap('helix2')
```

```
[56]: lonrho = grd.lon_rho.isel(xi_rho=slice(1,-1),eta_rho=slice(1,-1))
      latrho = grd.lat_rho.isel(xi_rho=slice(1,-1),eta_rho=slice(1,-1))

      fig,ax = plt.subplots(1,3,figsize=(12,3.5))

      # vorticity
      pc0=ax[0].pcolormesh(lonrho,latrho,Vort,vmin=-0.5,vmax=0.5,cmap=cmap_vrt)
```

```python
ax[0].annotate(r'$\zeta f^{-1}$',xy=(0.1,0.92),xycoords='axes fraction',ha='center',va='center',fontsize=12)

cax = fig.add_axes([ax[0].bbox.x0/fig.bbox.width+0.02,ax[0].bbox.y1/fig.bbox.height-0.2,
                    0.3*ax[0].bbox.width/fig.bbox.width,0.02])

cb0 = plt.colorbar(pc0,cax=cax,orientation='horizontal',extend='both')
cb0.set_ticks([-0.5,0,0.5]);cb0.set_ticklabels(['-0.5','0','0.5'],fontsize=11)
cb0.ax.xaxis.set_ticks_position('top')

# divergence
pc1=ax[1].pcolormesh(lonrho,latrho,Dive,vmin=-0.3,vmax=0.3,cmap=cmap_div)
ax[1].annotate(r'$\delta f^{-1}$',xy=(0.1,0.92),xycoords='axes fraction',ha='center',va='center',fontsize=12)

cax = fig.add_axes([ax[1].bbox.x0/fig.bbox.width+0.02,ax[1].bbox.y1/fig.bbox.height-0.2,
                    0.3*ax[1].bbox.width/fig.bbox.width,0.02])

cb1 = plt.colorbar(pc1,cax=cax,orientation='horizontal',extend='both')
cb1.set_ticks([-0.3,0,0.3]);cb1.set_ticklabels(['-0.3','0','0.3'],fontsize=11)
cb1.ax.xaxis.set_ticks_position('top')

# strain
pc2=ax[2].pcolormesh(lonrho,latrho,Stra,vmin=0,vmax=0.2,cmap=cmap_str)
ax[2].annotate(r'$\alpha f^{-1}$',xy=(0.1,0.92),xycoords='axes fraction',ha='center',va='center',fontsize=12)

cax = fig.add_axes([ax[2].bbox.x0/fig.bbox.width+0.02,ax[2].bbox.y1/fig.bbox.height-0.2,
                    0.3*ax[2].bbox.width/fig.bbox.width,0.02])

cb2 = plt.colorbar(pc2,cax=cax,orientation='horizontal',extend='both')
cb2.set_ticks([0,0.1,0.2]);cb2.set_ticklabels(['0','0.1','0.2'],fontsize=11)
cb2.ax.xaxis.set_ticks_position('top')
```



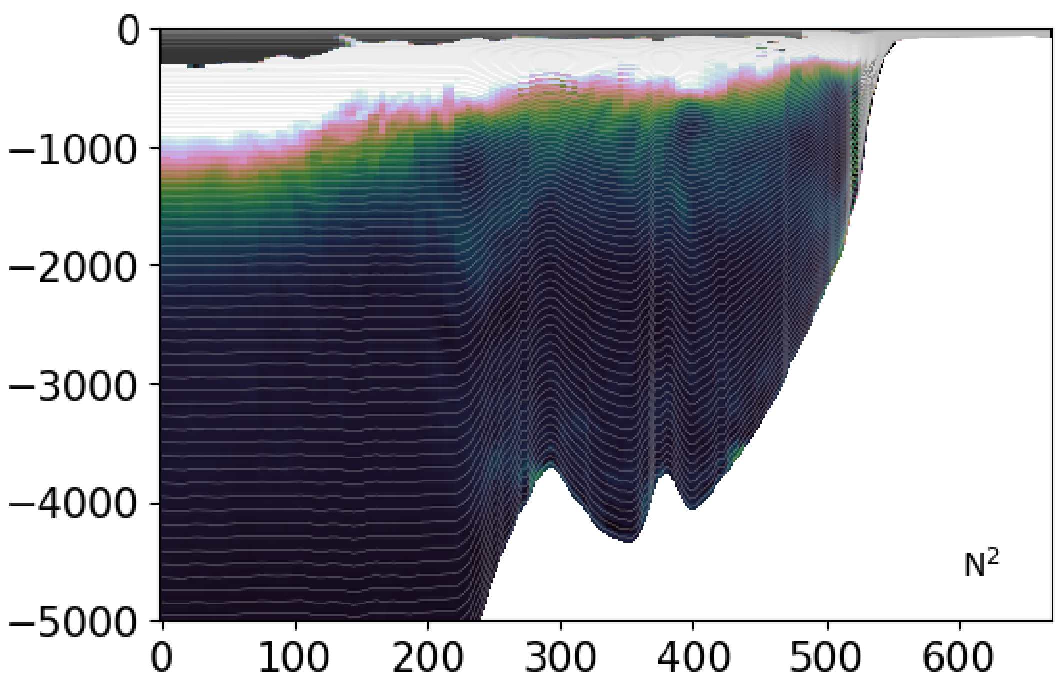
[57]:
```
N2 = -(9.81/ds.attrs['rho0'])*(diffz(ds.rho,zrho))
```

[58]:
```
# get vertical section
N2rad = get_vertical_section(N2,lon_rad2,lat_rad2)

zrad = get_vertical_section(zw,lon_rad2,lat_rad2)
zrad = zrad.isel(s_w=slice(1,-1))

DD = xr.ones_like(zrad)*N2rad.dist
```

[59]:
```
fig,ax=plt.subplots(1,1,figsize=(6,4))

ax.pcolormesh(DD,zrad,N2rad.T,cmap='cubehelix',vmin=1e-7,vmax=1e-5);
ax.plot(DD,zrad,lw=0.2,c='0.75');
ax.
 ↪annotate('N$^2$',xy=(617,-4500),xycoords='data',ha='center',va='center',fontsize=12)
ax.set_ylim(-5000,0)
```

```
/dev/shm/pbs.2350798.datarmor0/ipykernel_12072/403381004.py:3: UserWarning: The
input coordinates to pcolormesh are interpreted as cell centers, but are not
monotonically increasing or decreasing. This may lead to incorrectly calculated
cell edges, in which case, please supply explicit cell edges to pcolormesh.
  ax.pcolormesh(DD,zrad,N2rad.T,cmap='cubehelix',vmin=1e-7,vmax=1e-5);
```

[59]: (-5000.0, 0.0)



[60]:
```
# LOAD GIGATL1

path = '/home/datawork-lops-megatl/Crossroads/'

#␣
↪drop_variables=['time','time_instant','time_instant_bounds','time_counter_bounds']
gig1 = xr.open_dataset(path+'gigatl1_1h_tides_crossroads_1h_2008-04-01.
↪nc',engine='netcdf4',decode_cf=False)

# load and cut grid
grid_gig1 = xr.open_dataset('/home/datawork-lops-megatl/GIGATL1/gigatl1_grd.
↪nc'); #list(gig1.variables)

# Turn on chunking to activate dask and parallelize read/write.
chks=500
chunks = {'time':1,'xi_rho': chks,'eta_rho' :chks,'eta_v': chks, 'xi_u':␣
↪chks,'s_rho':1,'s_w':1}

gig1 = gig1.chunk(chunks)

grid_gig1 = grid_gig1.isel(xi_rho=slice(gig1.attrs['izoom0'],gig1.
↪attrs['izoom1']+1),eta_rho=slice(gig1.attrs['jzoom0'],gig1.
↪attrs['jzoom1']+1),
                           xi_u=slice(gig1.attrs['izoom0'],gig1.
↪attrs['izoom1']),eta_v=slice(gig1.attrs['jzoom0'],gig1.attrs['jzoom1']))
```

[61]:
```
gig1 = gig1.isel(time=0)
```

[62]:
```
# add grid and calculate z
gig1 = add_grd(gig1,grid_gig1)
zrho,_ = calc_vertical_coord(gig1)

# mask
zrho = zrho.where(zrho<=0)
```

---

calculate vorticity and T anomaly at the 1027.68 isobath

[63]:
```
zrho=zrho.chunk({'xi_rho':zrho.xi_rho.size,'eta_rho':zrho.eta_rho.size,'s_rho':
↪1})
zrho=zrho.compute()
```

[64]:
```
# add sigma0 density
```

```
gig1['rho'] =  rho_eos(gig1.temp,gig1.salt,zrho,9.81,gig1.
 ↪attrs['rho0'],z_w=None,sig0=True)
```

[65]:
```
%%time
iso=[1027.68]
u_iso    = interpolate_vertically(gig1.u,rho2u(gig1.
 ↪rho),iso,ztype='density',interp_boundaries=False)
v_iso    = interpolate_vertically(gig1.v,rho2v(gig1.
 ↪rho),iso,ztype='density',interp_boundaries=False)
```

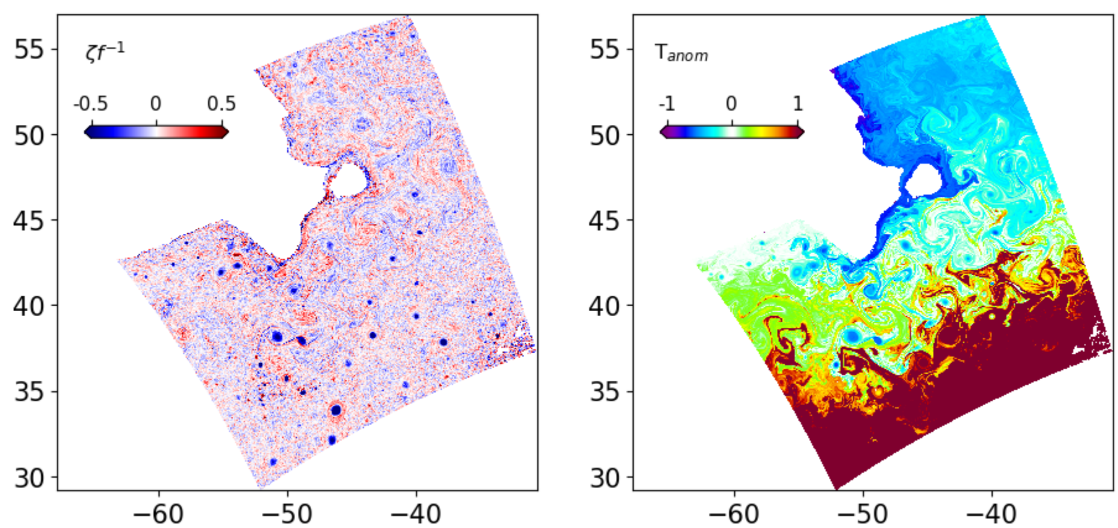
```
CPU times: user 712 ms, sys: 0 ns, total: 712 ms
Wall time: 708 ms
```

[66]:
```
%%time

# f0 [2D]
f0 = 2*7.29e-5*np.sin(np.deg2rad(gig1.lat_rho))

# calculate vorticity
Vort = psi2rho(diffx(v_iso.squeeze(),gig1.pm,grid='v',coords=False) -␣
 ↪diffy(u_iso.squeeze(),gig1.pn,grid='u',coords=False)) / f0
Vort = Vort.compute()

# calculate spatial temperature anomaly
temp_iso    = interpolate_vertically(gig1.temp,gig1.
 ↪rho,iso,ztype='density',interp_boundaries=False)

temp_anom= temp_iso - temp_iso.mean()
temp_anom = temp_anom.squeeze().compute()
```

```
CPU times: user 2min 9s, sys: 3.65 s, total: 2min 12s
Wall time: 3min 34s
```

[67]:
```
cmap_vrt=nc_colormap('blue_red')
cmap_anom=nc_colormap('jaison')

lonrho = gig1.lon_rho#.isel(xi_rho=slice(1,-1),eta_rho=slice(1,-1))
latrho = gig1.lat_rho#.isel(xi_rho=slice(1,-1),eta_rho=slice(1,-1))

fig,ax = plt.subplots(1,2,figsize=(11,5))

# vorticity
pc0=ax[0].pcolormesh(lonrho,latrho,Vort.T,vmin=-0.5,vmax=0.5,cmap=cmap_vrt)
ax[0].annotate(r'$\zeta f^{-1}$',xy=(0.1,0.92),xycoords='axes␣
 ↪fraction',ha='center',va='center',fontsize=12)
```

```
cax = fig.add_axes([ax[0].bbox.x0/fig.bbox.width+0.02,ax[0].bbox.y1/fig.bbox.
 ↪height-0.2,
                    0.3*ax[0].bbox.width/fig.bbox.width,0.02])

cb0 = plt.colorbar(pc0,cax=cax,orientation='horizontal',extend='both')
cb0.set_ticks([-0.5,0,0.5]);cb0.set_ticklabels(['-0.5','0','0.5'],fontsize=11)
cb0.ax.xaxis.set_ticks_position('top')

# temp anom
pc1=ax[1].pcolormesh(lonrho,latrho,temp_anom,vmin=-1,vmax=1,cmap=cmap_anom)
ax[1].annotate(r'T$_{anom}$',xy=(0.1,0.92),xycoords='axes␣
 ↪fraction',ha='center',va='center',fontsize=12)

cax = fig.add_axes([ax[1].bbox.x0/fig.bbox.width+0.02,ax[1].bbox.y1/fig.bbox.
 ↪height-0.2,
                    0.3*ax[1].bbox.width/fig.bbox.width,0.02])

cb1 = plt.colorbar(pc1,cax=cax,orientation='horizontal',extend='both')
cb1.set_ticks([-1,0,1]);cb1.set_ticklabels(['-1','0','1'],fontsize=11)
cb1.ax.xaxis.set_ticks_position('top')
```



[ ]: