nbdocs Documentation

Release 1.0

Roman Valov

Nov 18, 2020

Contents:

CHAPTER 1

Task 1

Hi, Erik. In this document I will guide you step-by-step into process of creating and deploying docs example to RTD. As you've requested the docs will have Jupyter Notebook.

1.1 Step 1. Overview

The RTD service is a document-hosting service for GitHub-hosted projects. It's free and it's only supposed to host documentation projects, not custom sites or files.

RTD is built on top of widely-used Sphinx project (https://www.sphinx-doc.org/en/master/). Sphinx is documentation generator, it produces documentation in various formats (primarily html) from templates files.

1.2 Step 2. Templates

Sphinx templates are just text files formatted using RST markup language. RST is similar to Markdown (markup language used in GitHub readme files or StackOverflow posts).

To get the idea of RST and learn basic constructs please read following doc:

https://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html

It's actually possible to use Markdown to write tempates for Sphinx however I've never used this possibility before.

1.3 Step 3. Installation

At first you have to install the Sphinx document generator on your system. If you're using Ubuntu run:

```
sudo apt-get install python3-sphinx
```

Alternatively you could use `pip3` command to install `sphinx` package from PyPI repository:

```
sudo pip3 install sphinx
```

Please note that there is python2 and python3 version of Sphinx could be available in your repository. You should prefer the same version as your primary project uses (Sphinx analyzes your python code to build documentation). Also please ensure you have only one version installed because you could get conflicts in ohter case.

Once installation completed, go to your source code directory and run:

```
sphinx-quickstart
```

This wizard will ask you several questions about your project and configuration options. It's safe to keep with defaults for all the configuration options, just ensure to set proper project name, author and version.

Take a minute to consider the structure of your sources. You should keep your *.ipynb files in the root of documentation.

1.4 Step 4. Build docs

When wizard completes you should find following files in your repository:

```
conf.py (configuration)
index.rst (home page template)
Makefile (build scripts for Linux)
make.bat (build scripts for Windows)
```

Directory could contain other files and subdirectories but they're not importrant.

In order to build html version of your docs run following command:

```
make html
```

This command will generate html docs in `_build/html` subdirectory (if you haven't changed the defaults) Now you can navigate with your file explorer to this directory and open `index.html` file in your browser. If you're on Ubuntu you could run following command without leaving the console:

```
xdg-open _build/html/index.html
```

1.5 Step 5. Notebooks extension

Hope everything is working fine and you were able to see your documentation stub in the browser. You could freely try various formatting constructs according to previously mentioned RST primer. But for the sake of simplicity I will continue to show you how to integrate Jupyter Notebooks.

Sphinx is extensible software and in order to render Jupyter Notebooks you have to install `nbsphinx` extension.

If you're on Ubuntu you should run:

```
sudo apt-get install python3-nbsphinx
```

Alternatively you could install PyPI version of the package:

```
sudo pip3 install nbsphinx
```

Now you should enable the extension in the `conf.py`. Open the file with your favorite editor and find `extensions` stanza. Add `'nbsphinx'` to the list of extensions, i.e.:

```
extensions = ['nbsphinx']
```

1.6 Step 6. Rendering notebooks

Now it's time to add your notebook to the docs. Make sure you have you *.ipynb file in the same directory with `index.rst`. Open `index.rst` with your favorite editor. By default the auto-generated file has Table of Contents and several standard links. You should modify table of contents and add the name of your *.ipynb file to the list. The .ipynb extension should be omitted. Also make sure your entry is idented on the same level as colon-marked stanzas:

```
.. toctree::
    :maxdepth: 2
    :caption: Contents:

    maps
```

As you see in my case I've added `maps` entry to the list. It's actually a copy of `python_maps_example.ipynb` from your repository renamed for the sake of convenience.

Once ready please run the build again and check results in your browser. Based on the `maps` file contents found in your repository you will get index page with pair of links on it. Each of the links will point to the sub-section in newly created `maps.html` file built from your notebook.

The same way you could freely use arbitrary *.ipynb file instead of RST-file, even instead of `index.rst`. However you have to delete `index.rst` file in latter case because *.rst files are prioritized by Sphinx.

1.7 Step 7. Moving to GitHub

If everything is working fine locally it's time to move to public hosting. In order to do that you should commit and upload your files to your GitHub repository.

The following files should be commited and pushed to the repository:

```
index.rst <your-noteook-file>.ipynb conf.py
```

As of `Makefile` and `make.bat` – they're just convenient wrappers for local builds and not required for RTD.

You could check that GitHub will render not only *.ipynb files in it's web-interface, but also *.rst files.

1.8 Step 8. Read the Docs

When your files are available on GitHub it's time to register an account on ReadTheDocs and link your GitHub repository.

Go to https://readthedocs.org/accounts/login/ and press the `Sign in with GitHub` button.

In the profile page of ReadTheDocs you will find `Import project` button, use it and select your repository from the list.

Once imported all the machinery should be set up by ReadTheDocs to start build and set up rebuild on each commit to your repo.

Please take a time to get familiar with ReadTheDocs interface.

In general it's usefull to be able to check the status of the last build and view the build logs.

1.9 Step 9. Notebooks on Read the Docs

By default ReadTheDocs is not configured to use Notebooks extension previously used for local build.

In order to change the limitations you have to add pair of configuration files to your repository.

At first, add the `requirements.txt` file to the same dir where you have `index.rst` located and add following lines:

```
ipykernel
nbsphinx
```

These lines will instruct ReadTheDocs build to download packages from the PyPI archive.

On your local setup `ipykernel` is usually installed as a dependency for Jupyter and `nbsphinx` was installed as a part of the tutorial.

At second, you have to add configuration file for ReadTheDocs service itself which relies on the `requirements.txt` defined. Configuration file for ReadTheDocs should be named `.readthedocs.yml` and should be located in top dir of your repository:

```
version: 2
formats: all
python:
  version: 3
  install:
  - requirements: docs/requirements.txt
  system_packages: true
```

As you see in my case the version of Python interpreter is set to 3 and `requirements.txt` is located in `docs` subdir.

Once files added do a commit and push to your repository, the ReadTheDocs will do the rebuild in a while.

1.10 Step 10. Themming

Sphinx supports themming. In my case Sphinx tools bundled with the distro are patched to use Alabaster theme by default.

In order to force your documentation pages to use particular theme it should be configured via `html_theme` parameter.

For example to use default ReadTheDocs theme you have to set `html_theme='sphinx_rtd_theme'` in your configuration file.

Being default for ReadTheDocs service it will be handled automatically on ReadTheDocs service. However if you wish to give it a try locally you have to install theme's python package:

```
sudo pip3 install sphinx-rtd-theme
```

[29]:
```
import numpy as np
import cartopy

import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
%matplotlib inline
```

CHAPTER 2

Point data

[32]:
```
import pandas as pd
```

[2]:
```
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/2011_
↪february_us_airport_traffic.csv')
```

[36]:
```
df.head()
```

[36]:
```
  iata                            airport               city state country  \
0  ORD       Chicago O'Hare International            Chicago    IL     USA
1  ATL  William B Hartsfield-Atlanta Intl            Atlanta    GA     USA
2  DFW    Dallas-Fort Worth International  Dallas-Fort Worth    TX     USA
3  PHX   Phoenix Sky Harbor International            Phoenix    AZ     USA
4  DEN                         Denver Intl             Denver    CO     USA

         lat        long    cnt
0  41.979595  -87.904464  25129
1  33.640444  -84.426944  21925
2  32.895951  -97.037200  20662
3  33.434167 -112.008056  17290
4  39.858408 -104.667002  13781
```

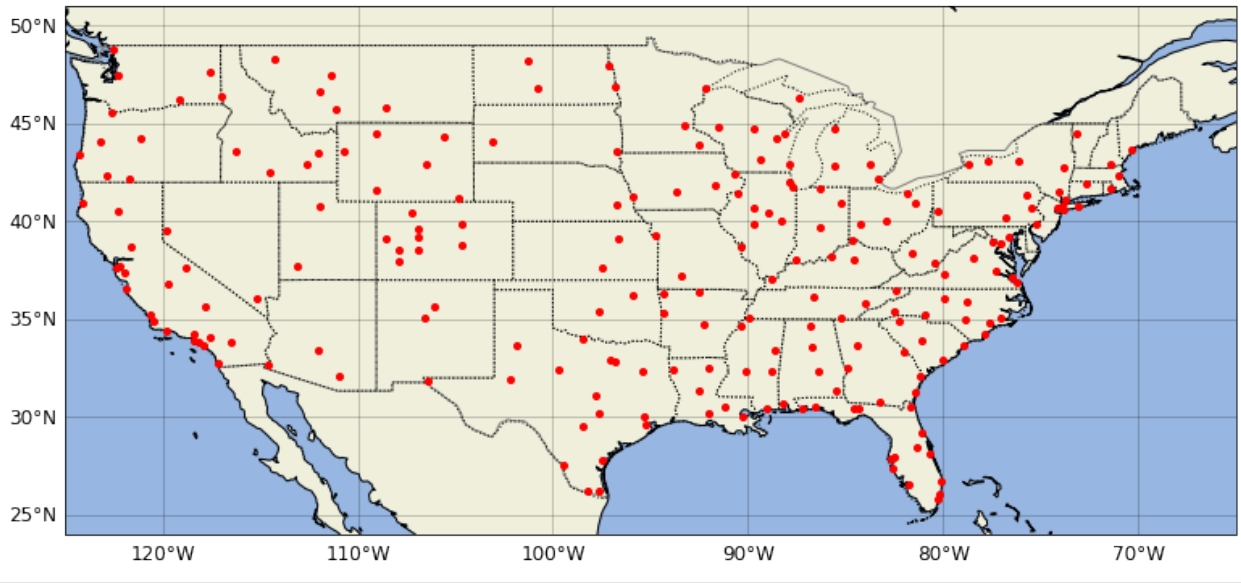
[35]:
```
plt.figure(figsize=(13,6.2))

ax = plt.axes(projection=cartopy.crs.PlateCarree())

# Set lat/lon limit of map
ax.set_extent([-125, -65, 24, 51], crs=cartopy.crs.PlateCarree())

# Add features
ax.add_feature(cartopy.feature.LAND)
ax.add_feature(cartopy.feature.OCEAN)
ax.add_feature(cartopy.feature.COASTLINE)
ax.add_feature(cartopy.feature.BORDERS, linestyle='-', color='grey')
```

(continues on next page)

(continued from previous page)

```
ax.add_geometries(cartopy.io.shapereader.Reader(cartopy.io.shapereader.natural_earth\
                                                (resolution='110m',category='cultural
↪',
                                                 name='admin_1_states_provinces_lakes_
↪shp')).geometries(),
                  cartopy.crs.PlateCarree(),facecolor='none',edgecolor='black',ls=':')

# Add lat/lon grid
gl = ax.gridlines(cartopy.crs.PlateCarree(), draw_labels=True, linewidth=1.0,␣
↪linestyle='-', color='k',alpha=0.2)
gl.xlocator = mticker.FixedLocator(np.arange(-120,-60,10))
gl.ylocator = mticker.FixedLocator(np.arange(25,55,5))
gl.top_labels = False
gl.right_labels = False
gl.xlabel_style= {'size': 12, 'color': 'k'}
gl.ylabel_style= {'size': 12, 'color': 'k'}

# Add airport locations
ax.plot(df.long, df.lat, transform=cartopy.crs.PlateCarree(),marker='o', color='red',␣
↪markersize=4, linestyle='')

plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/cartopy/mpl/style.py:76: UserWarning:␣
↪facecolor will have no effect as it has been defined as "never".
  warnings.warn('facecolor will have no effect as it has been '
```

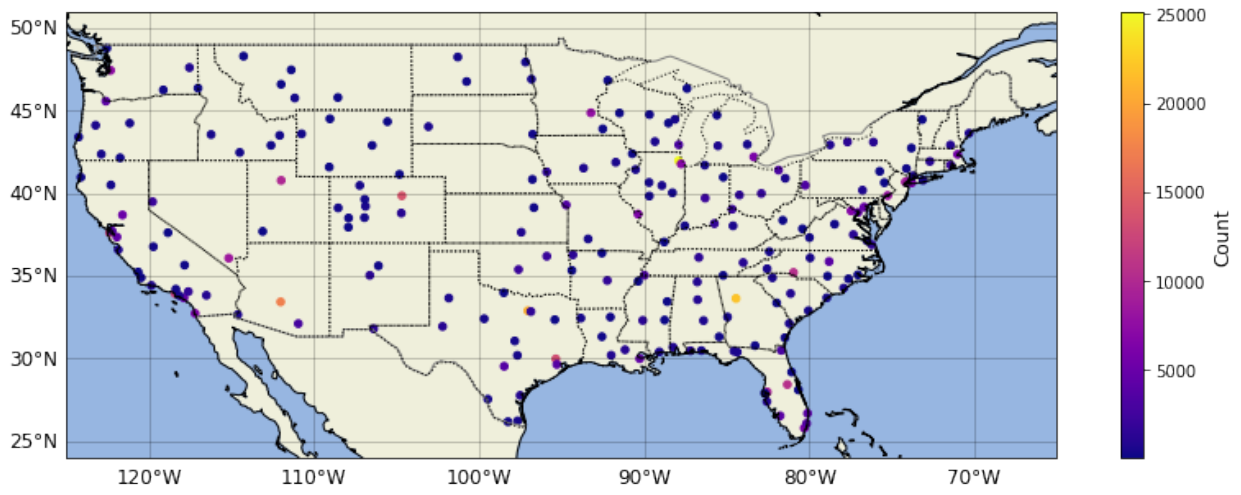


[43]:
```
fig = plt.figure(figsize=(14,5))

ax = plt.axes(projection=cartopy.crs.PlateCarree())

# Set lat/lon limit of map
ax.set_extent([-125, -65, 24, 51], crs=cartopy.crs.PlateCarree())

# Add features
ax.add_feature(cartopy.feature.LAND)
```

(continues on next page)

(continued from previous page)

```
ax.add_feature(cartopy.feature.OCEAN)
ax.add_feature(cartopy.feature.COASTLINE)
ax.add_feature(cartopy.feature.BORDERS, linestyle='-', color='grey')
ax.add_geometries(cartopy.io.shapereader.Reader(cartopy.io.shapereader.natural_earth\
                                                (resolution='110m',category='cultural
↪',
                                                 name='admin_1_states_provinces_lakes_
↪shp')).geometries(),
                  cartopy.crs.PlateCarree(),facecolor='none',edgecolor='black',ls=':')

# Add lat/lon grid
gl = ax.gridlines(cartopy.crs.PlateCarree(), draw_labels=True, linewidth=1.0,␣
↪linestyle='-', color='k',alpha=0.2)
gl.xlocator = mticker.FixedLocator(np.arange(-120,-60,10))
gl.ylocator = mticker.FixedLocator(np.arange(25,55,5))
gl.top_labels = False
gl.right_labels = False
gl.xlabel_style= {'size': 12, 'color': 'k'}
gl.ylabel_style= {'size': 12, 'color': 'k'}

# Add airport locations with color showing number of arrivals
p = ax.scatter(df.long, df.lat, c=df.cnt, transform=cartopy.crs.PlateCarree(), s = 20,
↪ cmap = 'plasma')
cb = fig.colorbar(p)
cb.set_label(r'Count',fontsize=12)
plt.show()
```



CHAPTER 3

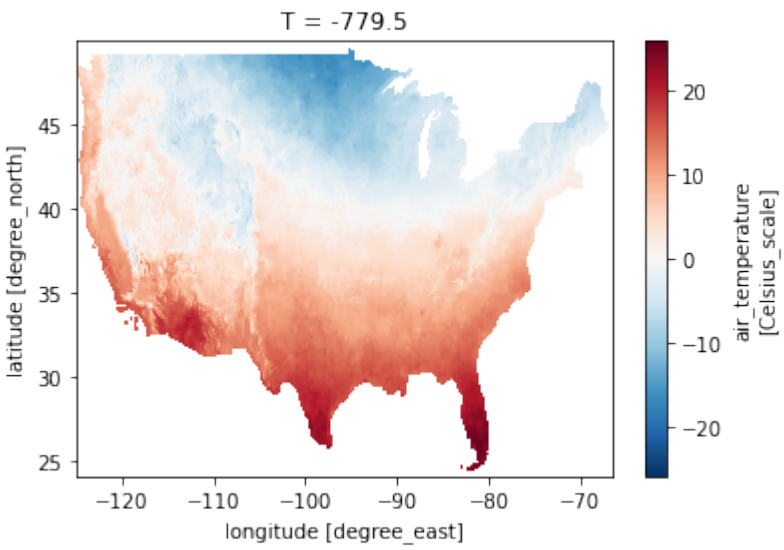
Gridded data

[53]:
```
import xarray as xr
```

[56]:
```
ds = xr.open_dataset("http://iridl.ldeo.columbia.edu/SOURCES/.OSU/.PRISM/.monthly/dods
↪",decode_times=False)
```

[59]:
```
ds.tmax[0].plot()
```

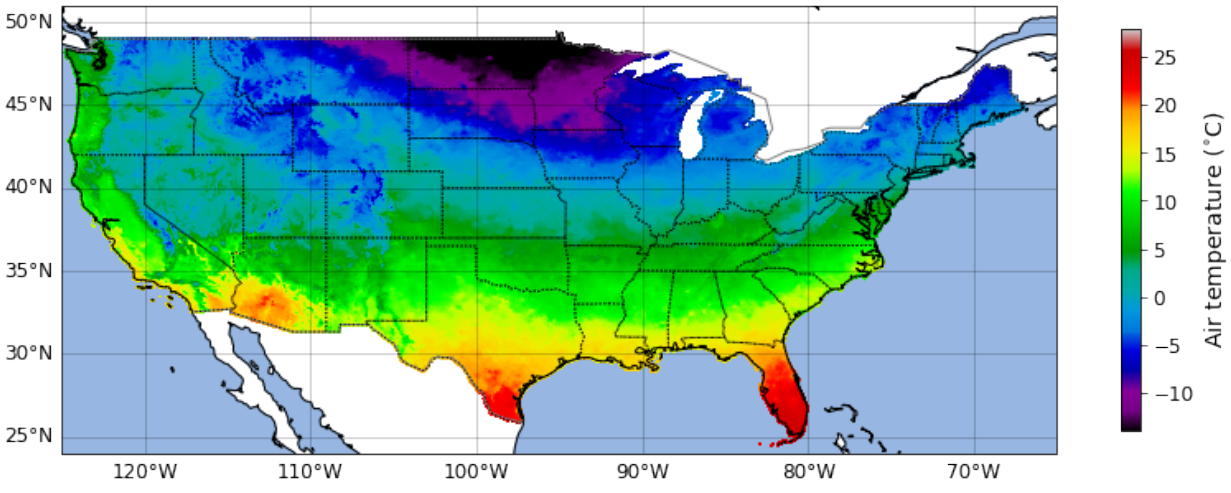
[59]:
```
<matplotlib.collections.QuadMesh at 0x7f2c0c9a51d0>
```



[71]:
```
dx = np.unique(np.round(np.diff(ds.X),4))[0]
dy = -np.unique(np.round(np.diff(ds.Y),4))[0]
print('Grid spacing\n',dx,dy)
```

```
Grid spacing
 0.0417 0.0417
```

[75]:
```
fig = plt.figure(figsize=(14,5))

ax = plt.axes(projection=cartopy.crs.PlateCarree())

# Set lat/lon limit of map
ax.set_extent([-125, -65, 24, 51], crs=cartopy.crs.PlateCarree())

# Add features
ax.add_feature(cartopy.feature.OCEAN)
ax.add_feature(cartopy.feature.COASTLINE)
ax.add_feature(cartopy.feature.BORDERS, linestyle='-', color='grey')
ax.add_geometries(cartopy.io.shapereader.Reader(cartopy.io.shapereader.natural_earth\
                                                (resolution='110m',category='cultural
↪',
                                                 name='admin_1_states_provinces_lakes_
↪shp')).geometries(),
                  cartopy.crs.PlateCarree(),facecolor='none',edgecolor='black',ls=':')

# Add lat/lon grid
gl = ax.gridlines(cartopy.crs.PlateCarree(), draw_labels=True, linewidth=1.0,␣
↪linestyle='-', color='k',alpha=0.2)
gl.xlocator = mticker.FixedLocator(np.arange(-120,-60,10))
gl.ylocator = mticker.FixedLocator(np.arange(25,55,5))
gl.top_labels = False
gl.right_labels = False
gl.xlabel_style= {'size': 12, 'color': 'k'}
gl.ylabel_style= {'size': 12, 'color': 'k'}

p = ax.pcolormesh(ds.X-dx/2, ds.Y-dy/2, np.ma.masked_invalid(ds.tmax[0]),
                  cmap='nipy_spectral', vmin=-14, vmax=28,transform=cartopy.crs.
↪PlateCarree())

cb = fig.colorbar(p, orientation='vertical', shrink=0.9, ticks=np.arange(-10,30,5))
cb.ax.tick_params(labelsize=12)
cb.set_label(r'Air temperature ($^{\circ}$C)', fontsize=14)
plt.show()
```



[ ]:

CHAPTER 4

Global Heat Budget Closure

Jan-Erik Tesdal[1,*], **Ryan Abernathey**[1] **and Ian Fenty**[2]

[1] Lamont-Doherty Earth Observatory, Columbia University, Palisades, NY, USA

[2] Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA

[*] *Corresponding author*: jt2796@columbia.edu

This section demonstrates the closure of the global heat budget in ECCOv4. The steps and Python code has been directly derived from the calculations and MATLAB code in *"A Note on Practical Evaluation of Budgets in ECCO Version 4 Release 3"* by Christopher G. Piecuch (https://ecco.jpl.nasa.gov/drive/files/Version4/Release3/doc/v4r3_budgets_howto.pdf).

4.1 Objectives

Evaluating and closing the heat budget over the global ocean.

4.2 Introduction

The ocean heat content (OHC) variability is described here with potential temperature (θ) which is given by the ECCOv4 diagnostic output `THETA`. The budget equation describing the change in θ is evaluated in general as

$$\frac{\partial \theta}{\partial t} = -\nabla \cdot (\theta \mathbf{u}) - \nabla \cdot \mathbf{F}^{\theta}_{\text{diff}} + F^{\theta}_{\text{forc}} \tag{4.1}$$

The heat budget includes the change in temperature over time ($\frac{\partial \theta}{\partial t}$), the convergence of heat advection ($-\nabla \cdot (\theta \mathbf{u})$) and heat diffusion ($-\nabla \cdot \mathbf{F}_{\text{diff}}$), plus downward heat flux from the atmosphere (F_{forc}). Note that in our definition F_{forc} contains both latent and sensible air-sea heat fluxes, longwave and shortwave radiation, as well as geothermal heat flux.

In the special case of ECCOv4, the heat budget is formulated as

$$\underbrace{\frac{\partial(s^*\theta)}{\partial t}}_{G^\theta_{\text{total}}} = \underbrace{-\nabla_{z^*}\cdot(s^*\theta\,\mathbf{v}_{res}) - \frac{\partial(\theta\,w_{res})}{\partial z^*}}_{G^\theta_{\text{advection}}} \underbrace{-s^*(\nabla\cdot\mathbf{F}^\theta_{\text{diff}})}_{G^\theta_{\text{diffusion}}} + \underbrace{s^*F^\theta_{\text{forc}}}_{G^\theta_{\text{forcing}}} \tag{4.2}$$

where $z^* = \frac{z-\eta}{H+\eta}H$ and $\nabla_{z^*}/\frac{\partial}{\partial z^*}$ are horizontal/vertical divergences in the z^* frame. Also note that the advection is now separated into horizontal ($\mathbf{v}_{res}$) and vertical (w_{res}) components, and there is a scaling factor ($s^* = 1 + \frac{\eta}{H}$) applied to the horizontal advection as well as the diffusion term ($G^\theta_{\text{diffusion}}$) and forcing term ($G^\theta_{\text{forcing}}$). s^* is a function of η which is the displacement of the ocean surface from its resting position of $z = 0$ (i.e., sea height anomaly). H is the ocean depth. s^* comes from the coordinate transformation from z to z^* (Campin and Adcroft, 2004; Campin et al., 2004). See ECCOv4 Global Volume Budget Closure for a more detailed explanation of the z^* coordinate system.

Note that the velocity terms in the ECCOv4 heat budget equation ($\mathbf{v}_{res}$ and w_{res}) are described as the "residual mean" velocities, which contain both the resolved (Eulerian) flow field, as well as the "GM bolus" velocity (i.e., parameterizing unresolved eddy effects):

$$(u_{res}, v_{res}, w_{res}) = (u, v, w) + (u_b, v_b, w_b)$$

Here (u_b, v_b, w_b) is the bolus velocity parameter, taking into account the correlation between velocity and thickness (also known as the eddy induced transportor the eddy advection term).

4.3 Evaluating the heat budget

We will evalute each term in the above heat budget

$$G^\theta_{\text{total}} = G^\theta_{\text{advection}} + G^\theta_{\text{diffusion}} + G^\theta_{\text{forcing}}$$

The total tendency of θ (G^θ_{total}) is the sum of the θ tendencies from advective heat convergence ($G^\theta_{\text{advection}}$), diffusive heat convergence ($G^\theta_{\text{diffusion}}$) and total forcing ($G^\theta_{\text{forcing}}$).

We present calculation sequentially for each term starting with G^θ_{total} which will be derived by differencing instantaneous monthly snapshots of θ. The terms on the right hand side of the heat budget are derived from monthly-averaged fields.

4.4 Prepare environment and load ECCOv4 diagnostic output

4.4.1 Import relevant Python modules

[1]:
```
import numpy as np
import xarray as xr
```

[2]:
```
# Suppress warning messages for a cleaner presentation
import warnings
warnings.filterwarnings('ignore')
```

[3]:
```
## Import the ecco_v4_py library into Python
## =========================================

## -- If ecco_v4_py is not installed in your local Python library,
##    tell Python where to find it.
```

(continues on next page)

(continued from previous page)

```
#import sys
#sys.path.append('/home/username/ECCOv4-py')

import ecco_v4_py as ecco
```

[4]:
```
# Plotting
import matplotlib.pyplot as plt
%matplotlib inline
```

4.4.2 Add relevant constants

[5]:
```
# Seawater density (kg/m^3)
rhoconst = 1029
## needed to convert surface mass fluxes to volume fluxes

# Heat capacity (J/kg/K)
c_p = 3994

# Constants for surface heat penetration (from Table 2 of Paulson and Simpson, 1977)
R = 0.62
zeta1 = 0.6
zeta2 = 20.0
```

4.4.3 Load ecco_grid

[6]:
```
## Set top-level file directory for the ECCO NetCDF files
## =====================================================================

# Define main directory
base_dir = '/home/username/ECCOv4-release'

# Define ECCO version
ecco_version = 'v4r3'

# Define a high-level directory for ECCO fields
ECCO_dir = base_dir + '/Release3_alt'
```

Note: Change `base_dir` to your own directory path.

[7]:
```
# Load the model grid
grid_dir= ECCO_dir + '/nctiles_grid/'
ecco_grid = ecco.load_ecco_grid_nc(grid_dir, 'ECCOv4r3_grid.nc')
```

4.4.4 Volume

Calculate the volume of each grid cell. This is used when converting advective and diffusive flux convergences and calculating volume-weighted averages.

[8]:
```
# Volume (m^3)
vol = (ecco_grid.rA*ecco_grid.drF*ecco_grid.hFacC).transpose('tile','k','j','i')
```

4.4.5 Load monthly snapshots

[9]:
```
data_dir= ECCO_dir + '/nctiles_monthly_snapshots'

year_start = 1993
year_end = 2017

# Load one extra year worth of snapshots
ecco_monthly_snaps = ecco.recursive_load_ecco_var_from_years_nc(data_dir, \
                         vars_to_load=['ETAN','THETA'],\
                         years_to_load=range(year_start, year_end+1))

num_months = len(ecco_monthly_snaps.time.values)
# Drop the last 11 months so that we have one snapshot at the beginning and end of␣
↪each month within the
# range 1993/1/1 to 2015/1/1

ecco_monthly_snaps = ecco_monthly_snaps.isel(time=np.arange(0, num_months-11))
```

```
loading files of  ETAN
loading files of  THETA
```

[10]:
```
# 1993-01 (beginning of first month) to 2015-01-01 (end of last month, 2014-12)
print(ecco_monthly_snaps.ETAN.time.isel(time=[0, -1]).values)
```

```
['1993-01-01T00:00:00.000000000' '2015-01-01T00:00:00.000000000']
```

[11]:
```
# Find the record of the last snapshot
## This is used to defined the exact period for monthly mean data
last_record_date = ecco.extract_yyyy_mm_dd_hh_mm_ss_from_datetime64(ecco_monthly_
↪snaps.time[-1].values)
print(last_record_date)
```

```
(2015, 1, 1, 0, 0, 0)
```

4.4.6 Load monthly mean data

[12]:
```
data_dir= ECCO_dir + '/nctiles_monthly'

year_end = last_record_date[0]
ecco_monthly_mean = ecco.recursive_load_ecco_var_from_years_nc(data_dir, \
                         vars_to_load=['TFLUX','oceQsw','ADVx_TH','ADVy_TH','ADVr_TH',
                                       'DFxE_TH','DFyE_TH','DFrE_TH','DFrI_TH'],\
                         years_to_load=range(year_start, year_end))
```

```
loading files of  ADVr_TH
loading files of  ADVx_TH
loading files of  ADVy_TH
loading files of  DFrE_TH
loading files of  DFrI_TH
loading files of  DFxE_TH
loading files of  DFyE_TH
loading files of  TFLUX
loading files of  oceQsw
```

[13]:
```
# Print first and last time points of the monthly-mean records
print(ecco_monthly_mean.time.isel(time=[0, -1]).values)
```

```
['1993-01-16T12:00:00.000000000' '2014-12-16T12:00:00.000000000']
```

Each monthly mean record is bookended by a snapshot. We should have one more snapshot than monthly mean record.

[14]:
```
print('Number of monthly mean records: ', len(ecco_monthly_mean.time))
print('Number of monthly snapshot records: ', len(ecco_monthly_snaps.time))
```

```
Number of monthly mean records:  264
Number of monthly snapshot records:  265
```

[15]:
```
# Drop superfluous coordinates (We already have them in ecco_grid)
ecco_monthly_mean = ecco_monthly_mean.reset_coords(drop=True)
```

4.4.7 Merge dataset of monthly mean and snapshots data

Merge the two datasets to put everything into one single dataset

[16]:
```
ds = xr.merge([ecco_monthly_mean,
               ecco_monthly_snaps.rename({'time':'time_snp','ETAN':'ETAN_snp', 'THETA
↪':'THETA_snp'})])
```

4.4.8 Create the xgcm 'grid' object

The `xgcm` 'grid' object is used to calculate the flux divergences across different tiles of the lat-lon-cap grid and the time derivatives from `THETA` snapshots

[17]:
```
# Change time axis of the snapshot variables
ds.time_snp.attrs['c_grid_axis_shift'] = 0.5
```

[18]:
```
grid = ecco.get_llc_grid(ds)
```

4.4.9 Number of seconds in each month

The xgcm `grid` object includes information on the time axis, such that we can use it to get Δt, which is the time span between the beginning and end of each month (in seconds).

[19]:
```
delta_t = grid.diff(ds.time_snp, 'T', boundary='fill', fill_value=np.nan)

# Convert to seconds
delta_t = delta_t.astype('f4') / 1e9
```

4.5 Calculate total tendency of θ ($G^{\theta}_{\text{total}}$)

We calculate the monthly-averaged time tendency of `THETA` by differencing monthly `THETA` snapshots. Remember that we need to include a scaling factor due to the nonlinear free surface formulation. Thus, we need to use snapshots of both `ETAN` and `THETA` to evaluate $s^*\theta$.

[20]:
```
# Calculate the s*theta term
sTHETA = ds.THETA_snp*(1+ds.ETAN_snp/ecco_grid.Depth)
```

[21]:
```
# Total tendency (psu/s)
G_total = grid.diff(sTHETA, 'T', boundary='fill', fill_value=0.0)/delta_t
```

Note: Unlike the monthly snapshots `ETAN_snp` and `THETA_snp`, the resulting data array `G_total` has now the same time values as the time-mean fields (middle of the month).

4.5.1 Plot the time-mean $\partial\theta/\partial t$, total $\Delta\theta$, and one example $\partial\theta/\partial t$ field

Time-mean $\partial\theta/\partial t$

The time-mean $\partial\theta/\partial t$ (i.e., $\overline{G^{\theta}_{\text{total}}}$), is given by

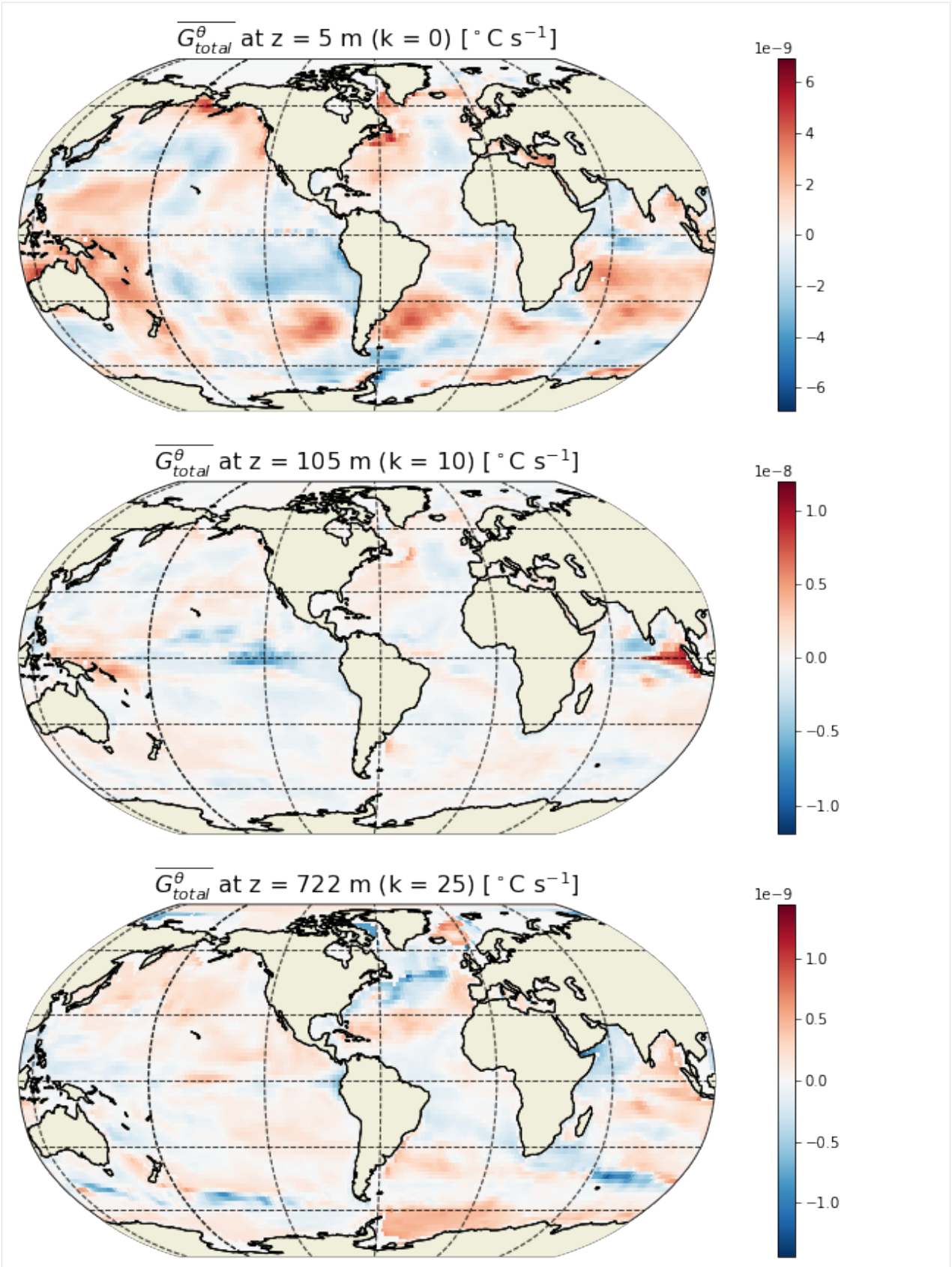
$\overline{G^{\theta}_{\text{total}}} = \sum_{i=1}^{nm} w_i G^{\theta}_{\text{total}}$

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

[22]:
```
# The weights are just the number of seconds per month divided by total seconds
month_length_weights = delta_t / delta_t.sum()
```

[23]:
```
# The weighted mean weights by the length of each month (in seconds)
G_total_mean = (G_total*month_length_weights).sum('time')
```

[24]:
```
plt.figure(figsize=(15,15))

for idx, k in enumerate([0,10,25]):
    p = ecco.plot_proj_to_latlon_grid(ecco_grid.XC, ecco_grid.YC, G_total_mean[:,k],
↪show_colorbar=True,
                                      cmap='RdBu_r', user_lon_0=-67, dx=2, dy=2,␣
↪subplot_grid=[3,1,idx+1]);
    p[1].set_title(r'$\overline{G^\theta_{total}}$ at z = %i m (k = %i) [$^\circ$C s$^
↪{-1}$]'\
                   %(np.round(-ecco_grid.Z[k].values),k), fontsize=16)
```


$\overline{G^{\theta}_{total}}$ at z = 5 m (k = 0) [$^{\circ}$C s^{-1}]
1e−9
6
4
2
0
−2
−4
−6
$\overline{G^{\theta}_{total}}$ at z = 105 m (k = 10) [$^{\circ}$C s^{-1}]
1e−8
1.0
0.5
0.0
−0.5
−1.0
$\overline{G^{\theta}_{total}}$ at z = 722 m (k = 25) [$^{\circ}$C s^{-1}]
1e−9
1.0
0.5
0.0
−0.5
−1.0

Total $\Delta\theta$

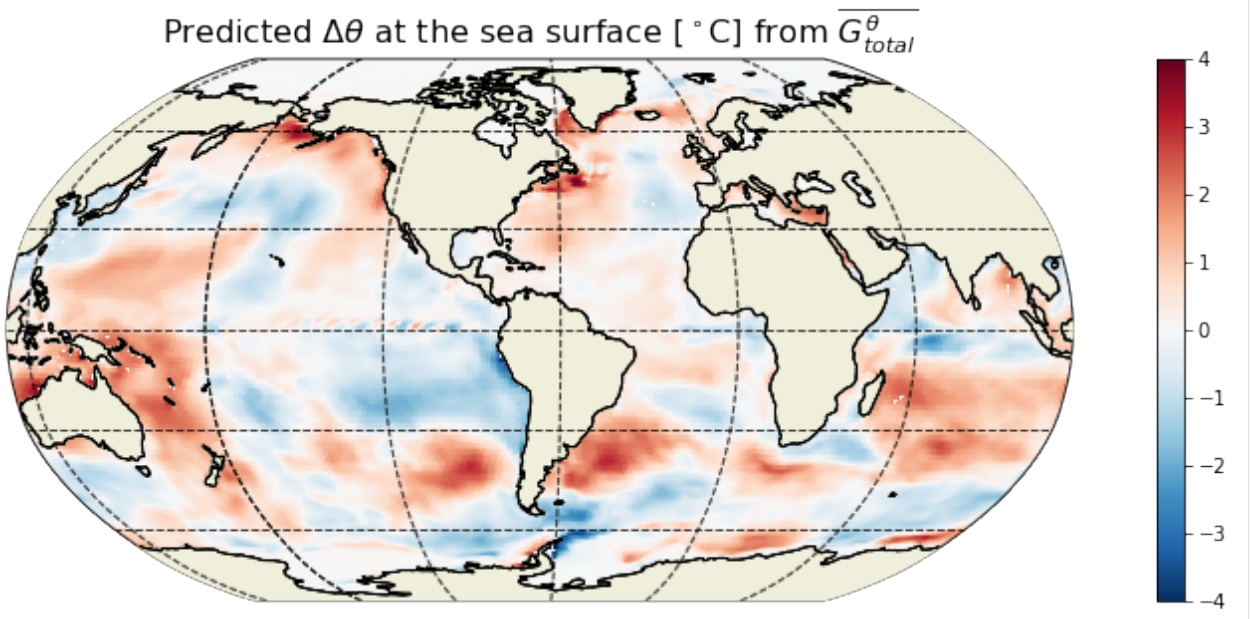
How much did THETA change over the analysis period?

[25]:
```
# The number of seconds in the entire period
seconds_in_entire_period = \
    float(ds.time_snp[-1] - ds.time_snp[0])/1e9
print ('seconds in analysis period: ', seconds_in_entire_period)

# which is also the sum of the number of seconds in each month
print('Sum of seconds in each month ', delta_t.sum().values)
```

```
seconds in analysis period:  694224000.0
Sum of seconds in each month  694224000.0
```

[26]:
```
THETA_delta = G_total_mean*seconds_in_entire_period
```

[27]:
```
plt.figure(figsize=(15,5));
ecco.plot_proj_to_latlon_grid(ecco_grid.XC, ecco_grid.YC, \
                              THETA_delta[:,0],show_colorbar=True,\
                              cmin=-4, cmax=4, \
                              cmap='RdBu_r', user_lon_0=-67, dx=0.2, dy=0.2);
plt.title(r'Predicted $\Delta \theta$ at the sea surface [$^\circ$C] from $\overline
↪{G^\theta_{total}}$',fontsize=16);
```



We can sanity check the total THETA change that we found by multipling the time-mean THETA tendency with the number of seconds in the simulation by comparing that with the difference in THETA between the end of the last month and start of the first month.

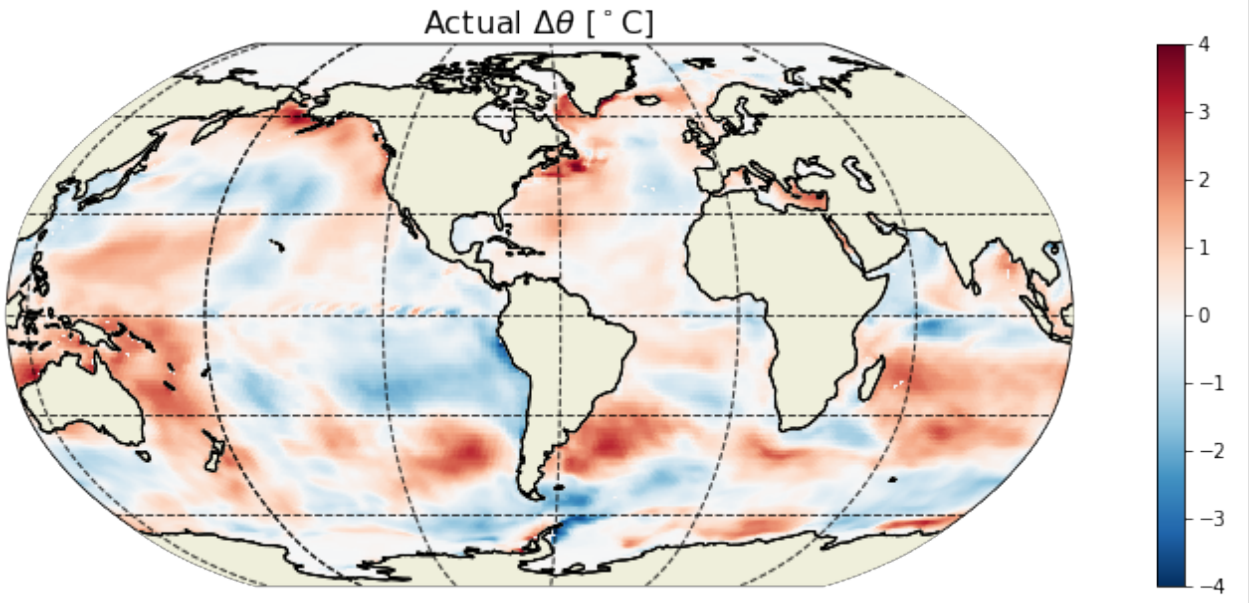
[28]:
```
THETA_delta_method_2 = ds.THETA_snp.isel(time_snp=-1) - ds.THETA_snp.isel(time_snp=0)
```

[29]:
```
plt.figure(figsize=(15,5));
ecco.plot_proj_to_latlon_grid(ecco_grid.XC, ecco_grid.YC, \
                              THETA_delta_method_2[:,0],show_colorbar=True,\
                              cmin=-4, cmax=4, \
```

(continues on next page)

(continued from previous page)

```
                                cmap='RdBu_r', user_lon_0=-67, dx=0.2, dy=0.2);
plt.title(r'Actual $\Delta \theta$ [$^\circ$C]', fontsize=16);
```

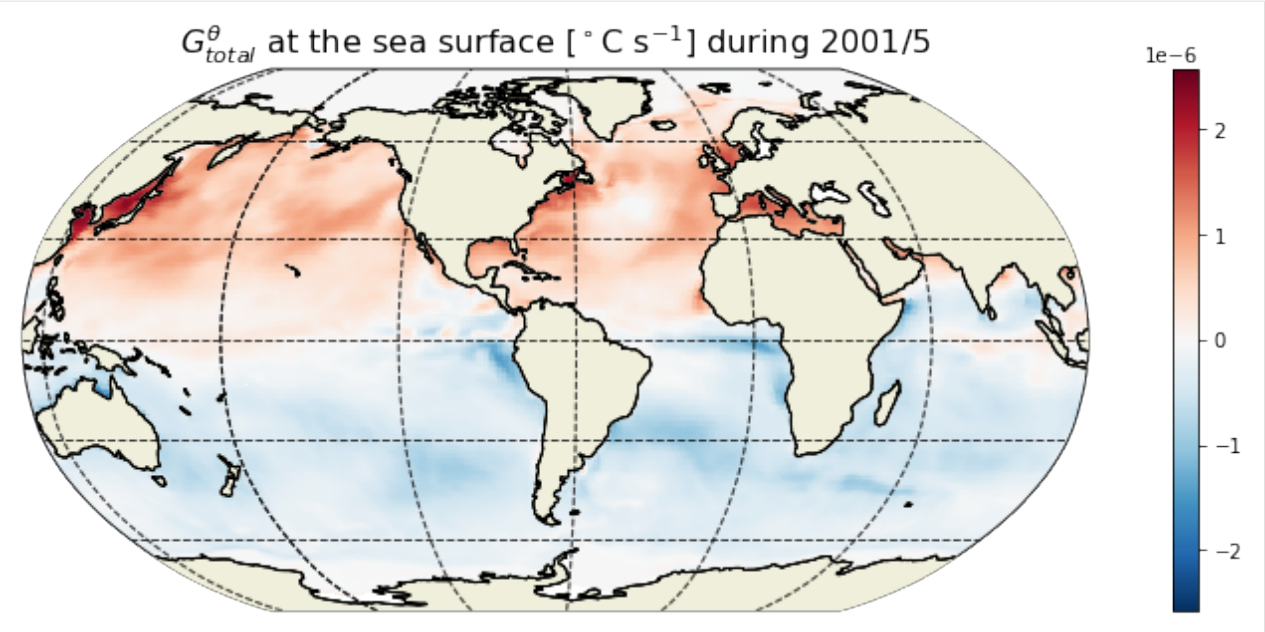


Example G^{θ}_{total} field at a particular time

[30]:
```
# get an array of YYYY, MM, DD, HH, MM, SS for
#dETAN_dT_perSec at time index 100
tmp = ecco.extract_yyyy_mm_dd_hh_mm_ss_from_datetime64(G_total.time[100].values)
print(tmp)
```

```
(2001, 5, 16, 12, 0, 0)
```

[31]:
```
plt.figure(figsize=(15,5));
ecco.plot_proj_to_latlon_grid(ecco_grid.XC, ecco_grid.YC, G_total.isel(time=100)[:,0],
↪ show_colorbar=True,
                                cmap='RdBu_r', user_lon_0=-67, dx=0.2, dy=0.2);

plt.title(r'$G^\theta_{total}$ at the sea surface [$^\circ$C s$^{-1}$] during ' +
          str(tmp[0]) +'/' + str(tmp[1]), fontsize=16);
```



For any given month the time rate of change of `THETA` is strongly dependent on the season. In the above we are looking at May 2001. We see positive `THETA` tendency in the northern hemisphere and cooling in the southern hemisphere.

4.6 Calculate tendency due to advective convergence ($G^{\theta}_{\text{advection}}$)

4.6.1 Horizontal convergence of advective heat flux

The relevant fields from the diagnostic output here are - `ADVx_TH`: U Component Advective Flux of Potential Temperature (degC m^3/s) - `ADVy_TH`: V Component Advective Flux of Potential Temperature (degC m^3/s)

The xgcm `grid` object is then used to take the convergence of the horizontal heat advection.

[32]:
```
ADVxy_diff = grid.diff_2d_vector({'X' : ds.ADVx_TH, 'Y' : ds.ADVy_TH}, boundary =
↪'fill')

# Convergence of horizontal advection (degC m^3/s)
adv_hConvH = (-(ADVxy_diff['X'] + ADVxy_diff['Y']))
```

4.6.2 Vertical convergence of advective heat flux

The relevant field from the diagnostic output is - `ADVr_TH`: Vertical Advective Flux of Potential Temperature (degC m^3/s)

[33]:
```
# Load monthly averages of vertical advective flux
ADVr_TH = ds.ADVr_TH.transpose('time','tile','k_l','j','i')
```

> **Note**: For `ADVr_TH`, `DFrE_TH` and `DFrI_TH`, we need to make sure that sequence of dimensions are consistent. When loading the fields use `.transpose('time','tile','k_l','j','i')`. Otherwise, the divergences will be not correct (at least for `tile = 12`).

[34]:
```
# Convergence of vertical advection (degC m^3/s)
adv_vConvH = grid.diff(ADVr_TH, 'Z', boundary='fill')
```

> **Note**: In case of the volume budget (and salinity conservation), the surface forcing (`oceFWflx`) is already included at the top level (`k_l = 0`) in `WVELMASS`. Thus, to keep the surface forcing term explicitly represented, one needs to zero out the values of `WVELMASS` at the surface so as to avoid double counting (see `ECCO_v4_Volume_budget_closure.ipynb`). This is not the case for the heat budget. `ADVr_TH` does not include the sea surface forcing. Thus, the vertical advective flux (at the air-sea interface) should not be zeroed out.

4.6.3 Total convergence of advective flux ($G^{\theta}_{\text{advection}}$)

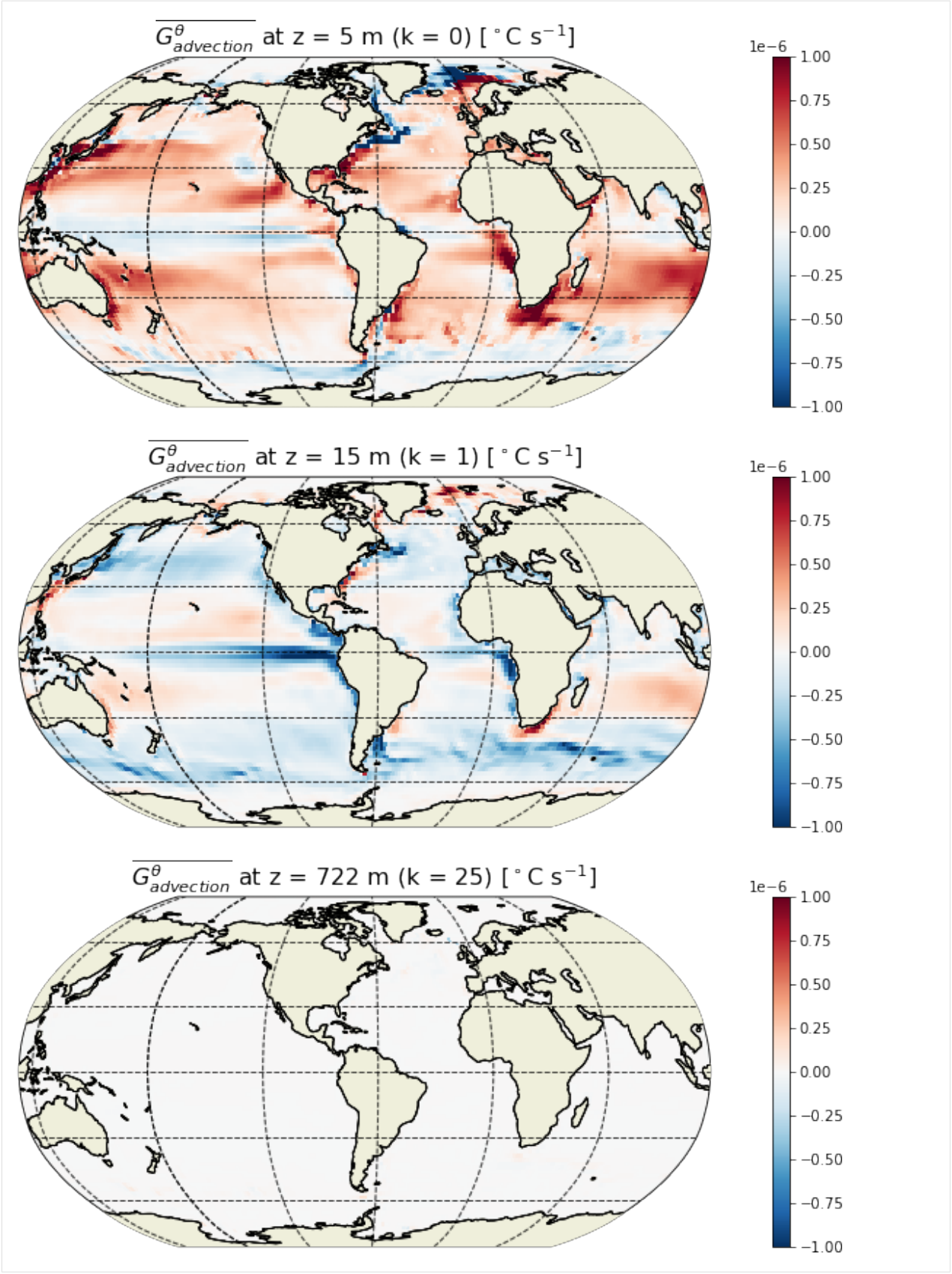
We can get the total convergence by simply adding the horizontal and vertical component.

[35]:
```
# Sum horizontal and vertical convergences and divide by volume (degC/s)
G_advection = (adv_hConvH + adv_vConvH)/vol
```

4.6.4 Plot the time-mean $G^{\theta}_{\text{advection}}$

[36]:
```
G_advection_mean = (G_advection*month_length_weights).sum('time')
```

[37]:
```
plt.figure(figsize=(15,15))

for idx, k in enumerate([0,1,25]):
    p = ecco.plot_proj_to_latlon_grid(ecco_grid.XC, ecco_grid.YC, G_advection_mean[:,
↪k],show_colorbar=True,
                                      cmin=-1e-6, cmax=1e-6, cmap='RdBu_r', user_lon_
↪0=-67, dx=2, dy=2,
                                      subplot_grid=[3,1,idx+1]);
    p[1].set_title(r'$\overline{G^\theta_{advection}}$ at z = %i m (k = %i) [$^\circ
↪$C s$^{-1}$]'\
                   %(np.round(-ecco_grid.Z[k].values),k), fontsize=16)
```

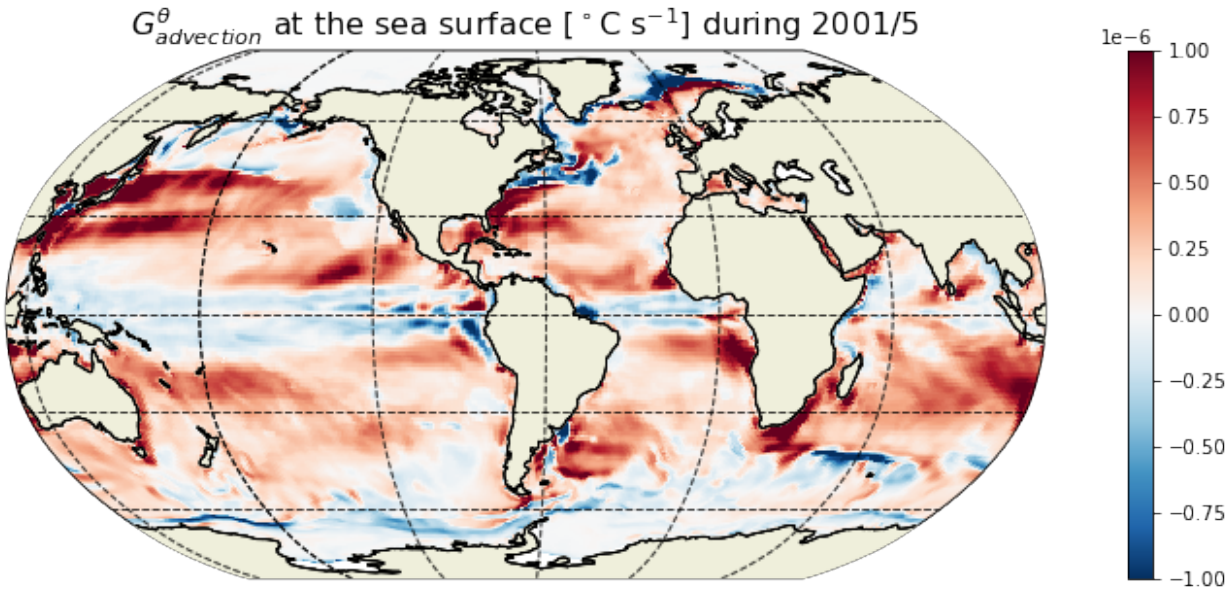

$\overline{G^{\theta}_{advection}}$ at z = 5 m (k = 0) [$^{\circ}$C s^{-1}]
1e−6
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
$\overline{G^{\theta}_{advection}}$ at z = 15 m (k = 1) [$^{\circ}$C s^{-1}]
1e−6
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
$\overline{G^{\theta}_{advection}}$ at z = 722 m (k = 25) [$^{\circ}$C s^{-1}]
1e−6
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

4.6.5 Example $G^{\theta}_{\text{advection}}$ field at a particular time

[38]:
```
tmp = ecco.extract_yyyy_mm_dd_hh_mm_ss_from_datetime64(G_advection.time[100].values)
print(tmp)
```

```
(2001, 5, 16, 12, 0, 0)
```

[39]:
```
plt.figure(figsize=(15,5));

ecco.plot_proj_to_latlon_grid(ecco_grid.XC, ecco_grid.YC, G_advection.isel(time=100)[:
↪,0], show_colorbar=True,
                                cmin=-1e-6, cmax=1e-6, cmap='RdBu_r', user_lon_0=-67,␣
↪dx=0.2, dy=0.2)
plt.title(r'$G^\theta_{advection}$ at the sea surface [$^\circ$C s$^{-1}$] during ' +
          str(tmp[0]) +'/' + str(tmp[1]), fontsize=16)
plt.show()
```



4.7 Calculate tendency due to diffusive convergence ($G^{\theta}_{\text{diffusion}}$)

4.7.1 Horizontal convergence of advective heat flux

The relevant fields from the diagnostic output here are - `DFxE_TH`: U Component Diffusive Flux of Potential Temperature (degC m^3/s) - `DFyE_TH`: V Component Diffusive Flux of Potential Temperature (degC m^3/s)

As with advective fluxes, we use the xgcm `grid` object to calculate the convergence of horizontal heat diffusion.

[40]:
```
DFxyE_diff = grid.diff_2d_vector({'X' : ds.DFxE_TH, 'Y' : ds.DFyE_TH}, boundary =
↪'fill')

# Convergence of horizontal diffusion (degC m^3/s)
dif_hConvH = (-(DFxyE_diff['X'] + DFxyE_diff['Y']))
```

4.7.2 Vertical convergence of advective heat flux

The relevant fields from the diagnostic output are - `DFrE_TH`: Vertical Diffusive Flux of Potential Temperature (Explicit part) (degC m^3/s) - `DFrI_TH`: Vertical Diffusive Flux of Potential Temperature (Implicit part) (degC m^3/s) > **Note**: Vertical diffusion has both an explicit (`DFrE_TH`) and an implicit (`DFrI_TH`) part.

[41]:
```
# Load monthly averages of vertical diffusive fluxes
DFrE_TH = ds.DFrE_TH.transpose('time','tile','k_l','j','i')
DFrI_TH = ds.DFrI_TH.transpose('time','tile','k_l','j','i')

# Convergence of vertical diffusion (degC m^3/s)
dif_vConvH = grid.diff(DFrE_TH, 'Z', boundary='fill') + grid.diff(DFrI_TH, 'Z',␣
↪boundary='fill')
```

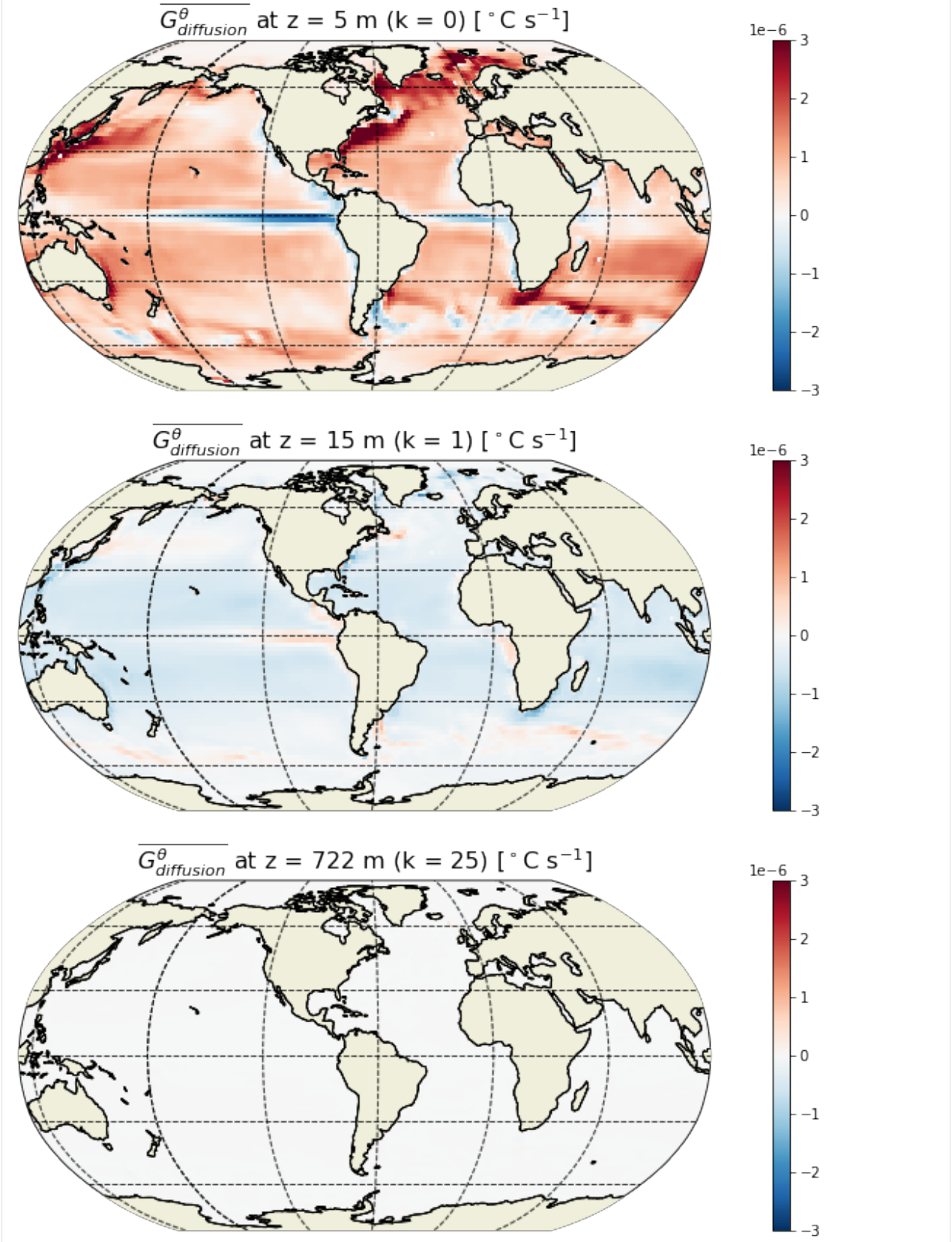
4.7.3 Total convergence of diffusive flux ($G^{\theta}_{\text{diffusion}}$)

[42]:
```
# Sum horizontal and vertical convergences and divide by volume (degC/s)
G_diffusion = (dif_hConvH + dif_vConvH)/vol
```

4.7.4 Plot the time-mean $G^{\theta}_{\text{diffusion}}$

[43]:
```
G_diffusion_mean = (G_diffusion*month_length_weights).sum('time')
```

[44]:
```
plt.figure(figsize=(15,15))

for idx, k in enumerate([0,1,25]):
    p = ecco.plot_proj_to_latlon_grid(ecco_grid.XC, ecco_grid.YC, G_diffusion_mean[:,
↪k],show_colorbar=True,
                                      cmin=-3e-6, cmax=3e-6, cmap='RdBu_r', user_lon_
↪0=-67, dx=2, dy=2,
                                      subplot_grid=[3,1,idx+1]);
    p[1].set_title(r'$\overline{G^\theta_{diffusion}}$ at z = %i m (k = %i) [$^\circ
↪$C s$^{-1}$]'\
                   %(np.round(-ecco_grid.Z[k].values),k), fontsize=16)
```

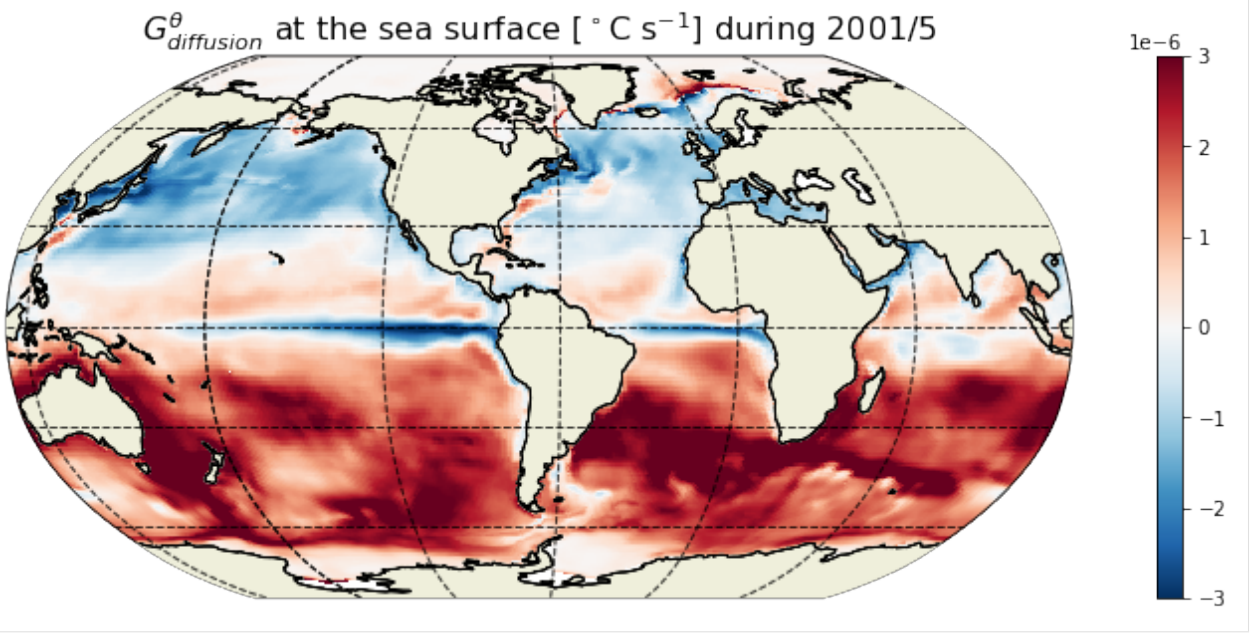

$\overline{G^{\theta}_{diffusion}}$ at z = 5 m (k = 0) [$^{\circ}$C s^{-1}]
1e−6
3
2
1
0
−1
−2
−3
$\overline{G^{\theta}_{diffusion}}$ at z = 15 m (k = 1) [$^{\circ}$C s^{-1}]
1e−6
3
2
1
0
−1
−2
−3
$\overline{G^{\theta}_{diffusion}}$ at z = 722 m (k = 25) [$^{\circ}$C s^{-1}]
1e−6
3
2
1
0
−1
−2
−3

4.7.5 Example $G^{\theta}_{\text{diffusion}}$ field at a particular time

[45]:
```
tmp = ecco.extract_yyyy_mm_dd_hh_mm_ss_from_datetime64(G_diffusion.time[100].values)
print(tmp)
```

```
(2001, 5, 16, 12, 0, 0)
```

[46]:
```
plt.figure(figsize=(15,5));

ecco.plot_proj_to_latlon_grid(ecco_grid.XC, ecco_grid.YC, G_diffusion.isel(time=100)[:
↪,0],show_colorbar=True,
                                cmin=-3e-6, cmax=3e-6, cmap='RdBu_r', user_lon_0=-67,␣
↪dx=0.2, dy=0.2)
plt.title(r'$G^\theta_{diffusion}$ at the sea surface [$^\circ$C s$^{-1}$] during ' +
          str(tmp[0]) +'/' + str(tmp[1]), fontsize=16)
plt.show()
```



4.8 Calculate tendency due to forcing ($G^{\theta}_{\text{forcing}}$)

Finally, we evaluate the local forcing term due to surface heat and geothermal fluxes.

4.8.1 Surface heat flux

For the surface contribution, there are two relevant model diagnostics: - `TFLUX`: total heat flux (match heat-content variations) (W/m^2) - `oceQsw`: net Short-Wave radiation (+=down) (W/m^2)

Defining terms needed for evaluating surface heat forcing

[47]:
```
Z = ecco_grid.Z.load()
RF = np.concatenate([ecco_grid.Zp1.values[:-1],[np.nan]])
```

Note: `Z` and `Zp1` are used in deriving surface heat penetration. MATLAB code uses `RF` from `mygrid` structure.

[48]:
```
q1 = R*np.exp(1.0/zeta1*RF[:-1]) + (1.0-R)*np.exp(1.0/zeta2*RF[:-1])
q2 = R*np.exp(1.0/zeta1*RF[1:]) + (1.0-R)*np.exp(1.0/zeta2*RF[1:])
```

[49]:
```
# Correction for the 200m cutoff
zCut = np.where(Z < -200)[0][0]
q1[zCut:] = 0
q2[zCut-1:] = 0
```

[50]:
```
# Save q1 and q2 as xarray data arrays
q1 = xr.DataArray(q1,coords=[Z.k],dims=['k'])
q2 = xr.DataArray(q2,coords=[Z.k],dims=['k'])
```

Compute vertically penetrating flux

Given the penetrating nature of the shortwave term, to properly evaluate the local forcing term, `oceQsw` must be removed from `TFLUX` (which contains the net latent, sensible, longwave, and shortwave contributions) and redistributed vertically.

[51]:
```
## Land masks
# Make copy of hFacC
mskC = ecco_grid.hFacC.copy(deep=True).load()

# Change all fractions (ocean) to 1. land = 0
mskC.values[mskC.values>0] = 1
```

[52]:
```
# Shortwave flux below the surface (W/m^2)
forcH_subsurf = ((q1*(mskC==1)-q2*(mskC.shift(k=-1)==1))*ds.oceQsw).transpose('time',
↪'tile','k','j','i')
```

[53]:
```
# Surface heat flux (W/m^2)
forcH_surf = ((ds.TFLUX - (1-(q1[0]-q2[0]))*ds.oceQsw)\
              *mskC[0]).transpose('time','tile','j','i').assign_coords(k=0).expand_
↪dims('k')
```

[54]:
```
# Full-depth sea surface forcing (W/m^2)
forcH = xr.concat([forcH_surf,forcH_subsurf[:,:,1:]], dim='k').transpose('time','tile
↪','k','j','i')
```

4.8.2 Geothermal flux

The geothermal flux contribution is not accounted for in any of the standard model diagnostics provided as output. Rather, this term, which is time invariant, is provided in the input file `geothermalFlux.bin` and can be downloaded from the PO.DAAC drive (https://ecco.jpl.nasa.gov/drive/files/Version4/Release3/input_init/geothermalFlux.bin). > **Note**: Here, `geothermalFlux.bin` has been placed in `base_dir`.

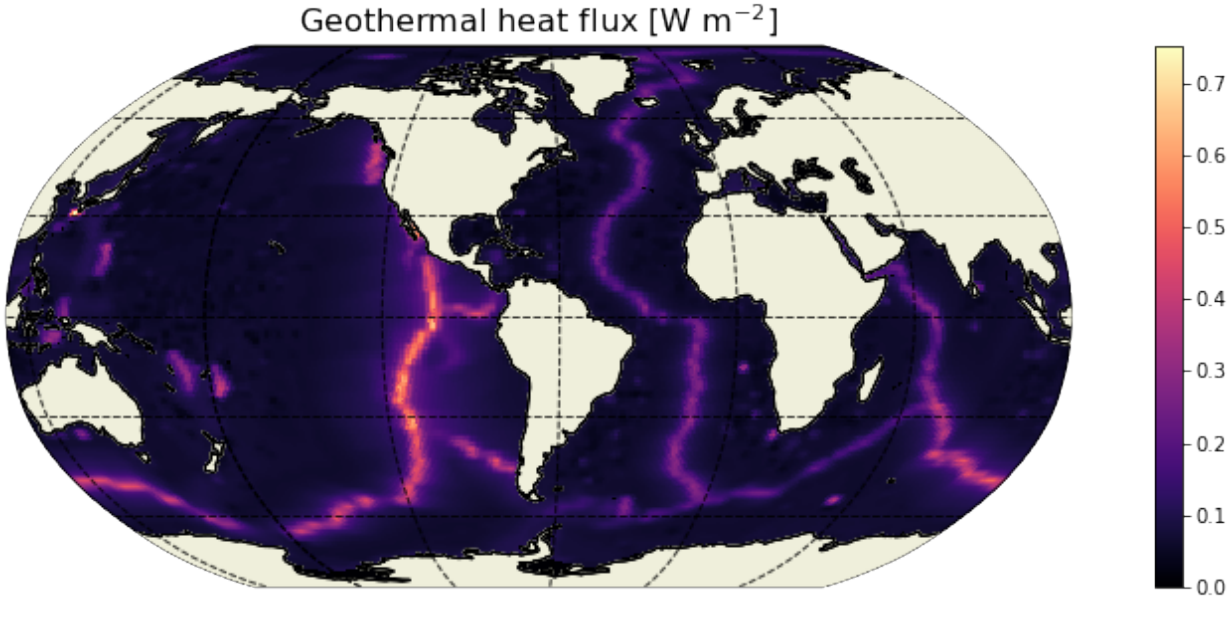
[55]:
```
# Load the geothermal heat flux using the routine 'read_llc_to_tiles'
geoflx = ecco.read_llc_to_tiles(base_dir, 'geothermalFlux.bin')
```

```
load_binary_array: loading file /work/noaa/gfdlscr/jtesdal/ECCOv4-release/
↪geothermalFlux.bin
load_binary_array: data array shape  (1170, 90)
load_binary_array: data array type  >f4
llc_compact_to_faces: dims, llc  (1170, 90) 90
llc_compact_to_faces: data_compact array type  >f4
llc_faces_to_tiles: data_tiles shape  (13, 90, 90)
llc_faces_to_tiles: data_tiles dtype  >f4
```

The geothermal flux dataset needs to be saved as an xarray data array with the same format as the model output.

[56]:
```
# Convert numpy array to an xarray DataArray with matching dimensions as the monthly␣
↪mean fields
geoflx_llc = xr.DataArray(geoflx,coords={'tile': ecco_monthly_mean.tile.values,
                                         'j': ecco_monthly_mean.j.values,
                                         'i': ecco_monthly_mean.i.values},dims=['tile
↪','j','i'])
```

[57]:
```
plt.figure(figsize=(15,5));

ecco.plot_proj_to_latlon_grid(ecco_grid.XC, ecco_grid.YC, geoflx_llc,show_
↪colorbar=True,cmap='magma',
                              user_lon_0=-67, dx=0.2, dy=0.2)
plt.title(r'Geothermal heat flux [W m$^{-2}$]', fontsize=16)
plt.show()
```



Geothermal flux needs to be a three dimensional field since the sources are distributed along the ocean floor at various depths. This requires a three dimensional mask.

[58]:
```
# Create 3d bathymetry mask
mskC_shifted = mskC.shift(k=-1)

mskC_shifted.values[-1,:,:,:] = 0
mskb = mskC - mskC_shifted
```

(continues on next page)

(continued from previous page)

```
# Create 3d field of geothermal heat flux
geoflx3d = geoflx_llc * mskb.transpose('k','tile','j','i')
GEOFLX = geoflx3d.transpose('k','tile','j','i')
GEOFLX.attrs = {'standard_name': 'GEOFLX','long_name': 'Geothermal heat flux','units':
↪ 'W/m^2'}
```

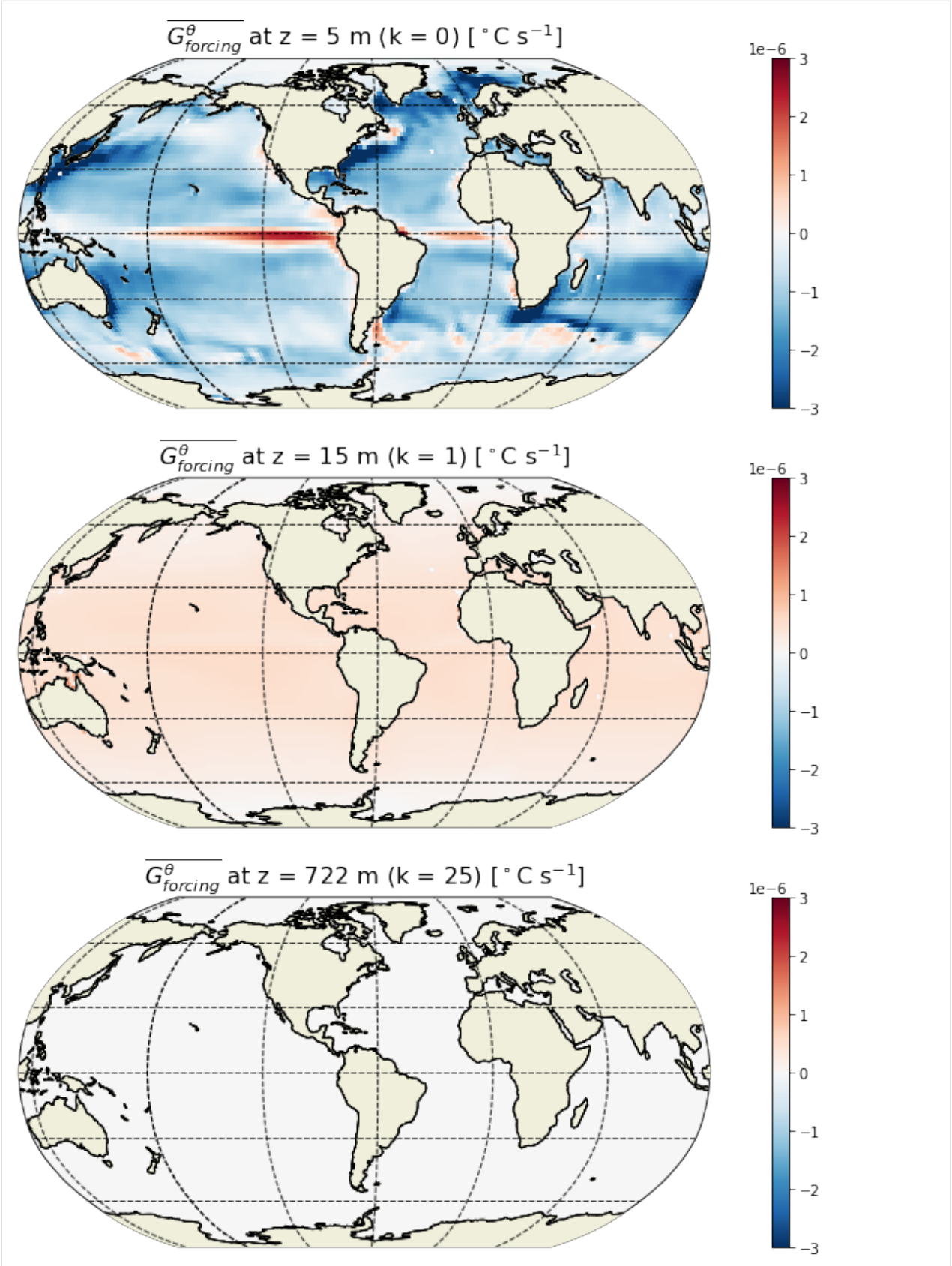
4.8.3 Total forcing ($G^{\theta}_{\text{forcing}}$)

[59]:
```
# Add geothermal heat flux to forcing field and convert from W/m^2 to degC/s
G_forcing = ((forcH + GEOFLX)/(rhoconst*c_p))/(ecco_grid.hFacC*ecco_grid.drF)
```

4.8.4 Plot the time-mean $G^{\theta}_{\text{forcing}}$

[60]:
```
G_forcing_mean = (G_forcing*month_length_weights).sum('time')
```

[61]:
```
plt.figure(figsize=(15,15))

for idx, k in enumerate([0,1,25]):
    p = ecco.plot_proj_to_latlon_grid(ecco_grid.XC, ecco_grid.YC, G_forcing_mean[:,k],
↪show_colorbar=True,
                                      cmin=-3e-6, cmax=3e-6, cmap='RdBu_r', user_lon_
↪0=-67, dx=2, dy=2,
                                      subplot_grid=[3,1,idx+1]);
    p[1].set_title(r'$\overline{G^\theta_{forcing}}$ at z = %i m (k = %i) [$^\circ$C s
↪$^{-1}$]'\
                   %(np.round(-ecco_grid.Z[k].values),k), fontsize=16)
```


$\overline{G^{\theta}_{forcing}}$ at z = 5 m (k = 0) [$^{\circ}$C s^{-1}]
1e−6
3
2
1
0
−1
−2
−3
$\overline{G^{\theta}_{forcing}}$ at z = 15 m (k = 1) [$^{\circ}$C s^{-1}]
1e−6
3
2
1
0
−1
−2
−3
$\overline{G^{\theta}_{forcing}}$ at z = 722 m (k = 25) [$^{\circ}$C s^{-1}]
1e−6
3
2
1
0
−1
−2
−3

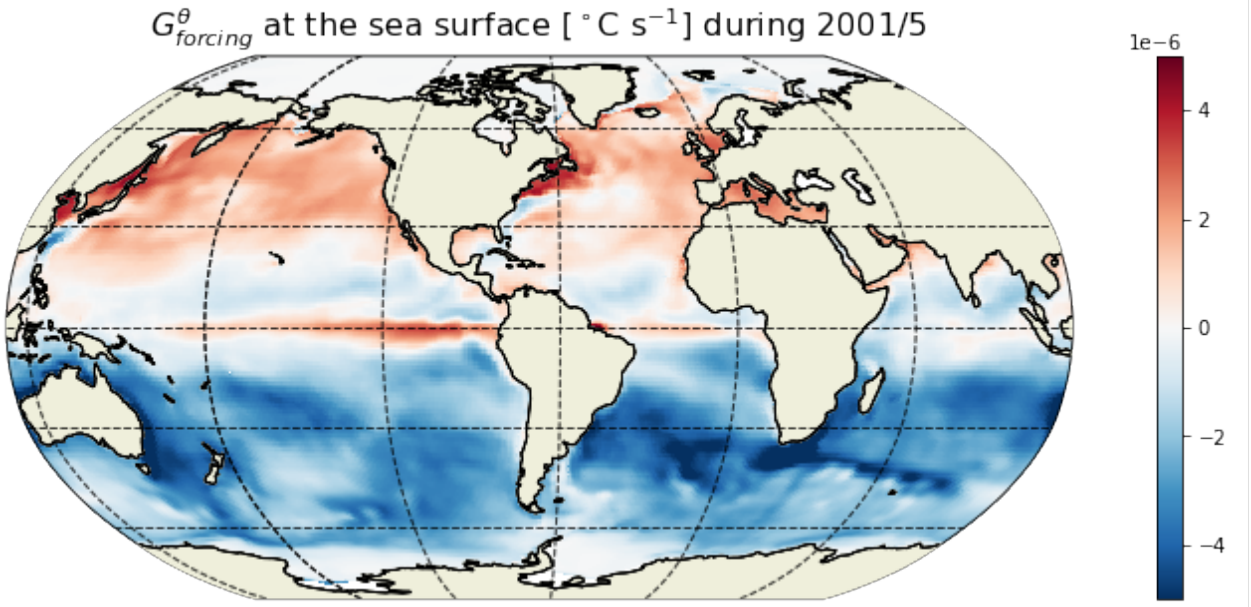
$\overline{G^{\theta}_{forcing}}$ is focused at the sea surface and much smaller (essentially zero) at depth. $\overline{G^{\theta}_{forcing}}$ is negative for most of the ocean (away from the equator). The spatial pattern in the surface forcing is the same as for diffusion but with opposite sign (see maps for $\overline{G^{\theta}_{diffusion}}$ above). This makes sense as forcing is to a large extent balanced by diffusion within the mixed layer.

4.8.5 Example $G^{\theta}_{\mathbf{forcing}}$ field at a particular time

[62]:
```
tmp = ecco.extract_yyyy_mm_dd_hh_mm_ss_from_datetime64(G_forcing.time[100].values)
print(tmp)
```

```
(2001, 5, 16, 12, 0, 0)
```

[63]:
```
plt.figure(figsize=(15,5));

ecco.plot_proj_to_latlon_grid(ecco_grid.XC, ecco_grid.YC, G_forcing.isel(time=100)[:,
↪0],show_colorbar=True,
                              cmin=-5e-6, cmax=5e-6, cmap='RdBu_r', user_lon_0=-67,␣
↪dx=0.2, dy=0.2)
plt.title(r'$G^\theta_{forcing}$ at the sea surface [$^\circ$C s$^{-1}$] during ' +
          str(tmp[0]) +'/' + str(tmp[1]), fontsize=16)
plt.show()
```



4.9 Save to dataset

Now that we have all the terms evaluated, let's save them to a dataset. Here are two examples: - Zarr is a new format that is used for cloud storage. - Netcdf is the more traditional format that most people are familiar with.

4.9.1 Add all variables to a new dataset

[65]:
```
varnames = ['G_total','G_advection','G_diffusion','G_forcing']

ds = xr.Dataset(data_vars={})
for varname in varnames:
    ds[varname] = globals()[varname].chunk(chunks={'time':1,'tile':13,'k':50,'j':90,'i
↪':90})
```

[66]:
```
# Add surface forcing (degC/s)
ds['Qnet'] = ((forcH /(rhoconst*c_p))\
              /(ecco_grid.hFacC*ecco_grid.drF)).chunk(chunks={'time':1,'tile':13,'k':
↪50,'j':90,'i':90})
```

[67]:
```
# Add shortwave penetrative flux (degC/s)
#Since we only are interested in the subsurface heat flux we need to zero out the top␣
↪cell
SWpen = ((forcH_subsurf /(rhoconst*c_p))/(ecco_grid.hFacC*ecco_grid.drF)).where(forcH_
↪subsurf.k>0).fillna(0.)
ds['SWpen'] = SWpen.where(ecco_grid.hFacC>0).chunk(chunks={'time':1,'tile':13,'k':50,
↪'j':90,'i':90})
```

Note: `Qnet` and `SWpen` are included in `G_forcing` and are not necessary to close the heat budget.

[68]:
```
ds.time.encoding = {}
ds = ds.reset_coords(drop=True)
```

4.9.2 Save to zarr

[69]:
```
from dask.diagnostics import ProgressBar
```

[70]:
```
with ProgressBar():
    ds.to_zarr(base_dir + '/eccov4r3_budg_heat')
```

```
[########################################] | 100% Completed |  2min 40.7s
```

4.9.3 Save to netcdf

[70]:
```
with ProgressBar():
    ds.to_netcdf(base_dir + '/eccov4r3_budg_heat.nc', format='NETCDF4')
```

```
[########################################] | 100% Completed | 14min 17.5s
```

4.10 Load budget variables from file

After having saved the budget terms to file, we can load the dataset like this

[64]:
```
# Load terms from zarr dataset
G_total = xr.open_zarr(base_dir + '/eccov4r3_budg_heat').G_total
G_advection = xr.open_zarr(base_dir + '/eccov4r3_budg_heat').G_advection
G_diffusion = xr.open_zarr(base_dir + '/eccov4r3_budg_heat').G_diffusion
G_forcing = xr.open_zarr(base_dir + '/eccov4r3_budg_heat').G_forcing
```

(continues on next page)

(continued from previous page)

```
Qnet = xr.open_zarr(base_dir + '/eccov4r3_budg_heat').Qnet
SWpen = xr.open_zarr(base_dir + '/eccov4r3_budg_heat').SWpen
```

Or if you saved it as a netcdf file:

```
# Load terms from netcdf file
G_total_tendency = xr.open_dataset(base_dir + '/eccov4r3_budg_heat.nc').G_total_
↪tendency
G_advection = xr.open_dataset(base_dir + '/eccov4r3_budg_heat.nc').G_advection
G_diffusion = xr.open_dataset(base_dir + '/eccov4r3_budg_heat.nc').G_diffusion
G_forcing = xr.open_dataset(base_dir + '/eccov4r3_budg_heat.nc').G_forcing
Qnet = xr.open_dataset(base_dir + '/eccov4r3_budg_heat.nc').Qnet
```

4.11 Comparison between LHS and RHS of the budget equation

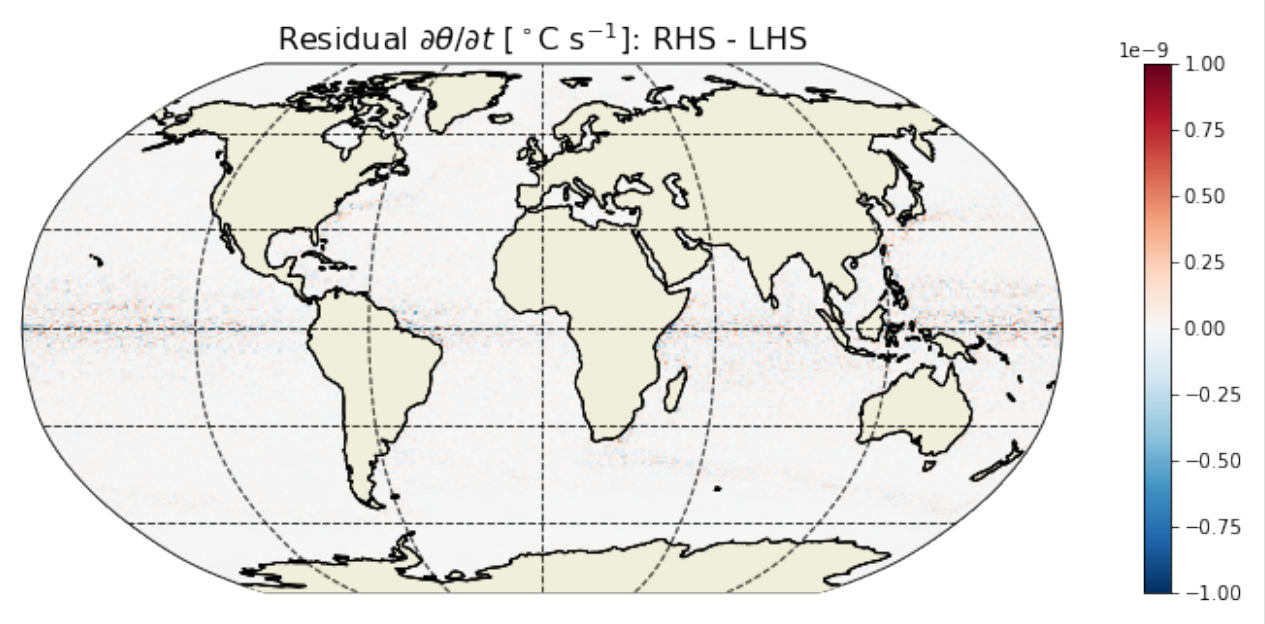
[65]:
```
# Total convergence
ConvH = G_advection + G_diffusion
```

[66]:
```
# Sum of terms in RHS of equation
rhs = ConvH + G_forcing
```

4.11.1 Map of residuals

[67]:
```
res = (rhs-G_total).sum(dim='k').sum(dim='time').compute()
```

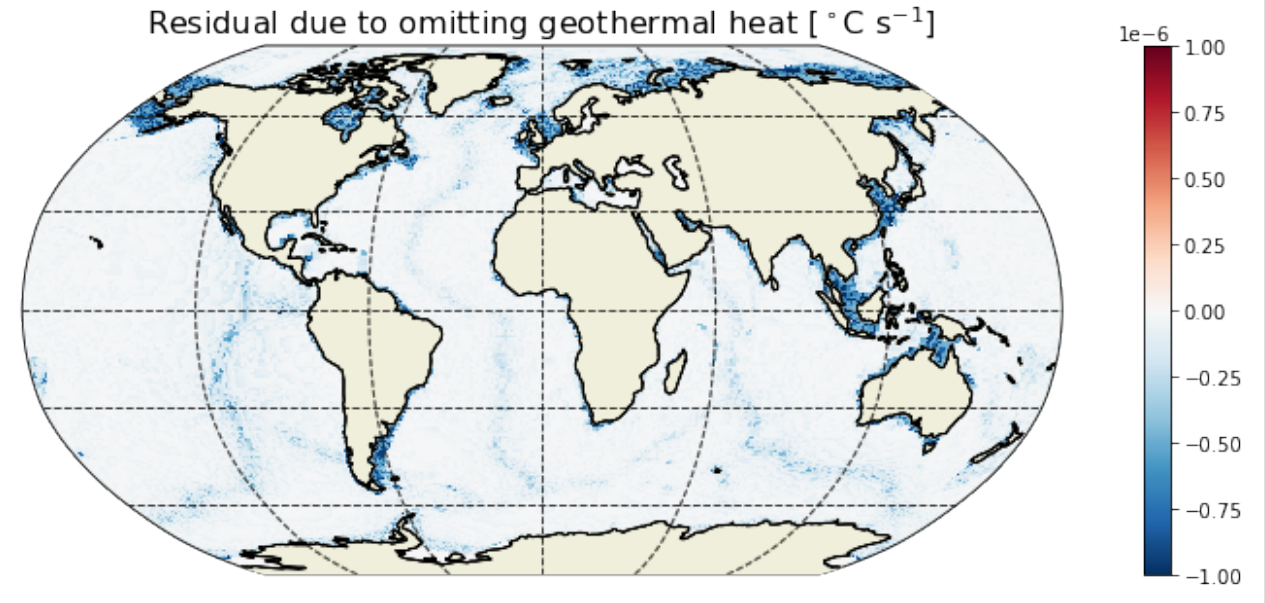
[68]:
```
plt.figure(figsize=(15,5))
ecco.plot_proj_to_latlon_grid(ecco_grid.XC, ecco_grid.YC, res,
                              cmin=-1e-9, cmax=1e-9, show_colorbar=True, cmap='RdBu_r
↪',dx=0.2, dy=0.2)
plt.title(r'Residual $\partial \theta / \partial t$ [$^\circ$C s$^{-1}$]: RHS - LHS',␣
↪fontsize=16)
plt.show()
```



The residual (summed over depth and time) is essentially zero everywhere. What if we omit the geothermal heat flux?

[69]:
```
# Residual when omitting geothermal heat flux
res_geo = (ConvH + Qnet - G_total).sum(dim='k').sum(dim='time').compute()
```

[70]:
```
plt.figure(figsize=(15,5))
ecco.plot_proj_to_latlon_grid(ecco_grid.XC, ecco_grid.YC, res_geo,
                              cmin=-1e-6, cmax=1e-6, show_colorbar=True, cmap='RdBu_r
↪', dx=0.2, dy=0.2)
plt.title(r'Residual due to omitting geothermal heat [$^\circ$C s$^{-1}$] ',␣
↪fontsize=16)
plt.show()
```

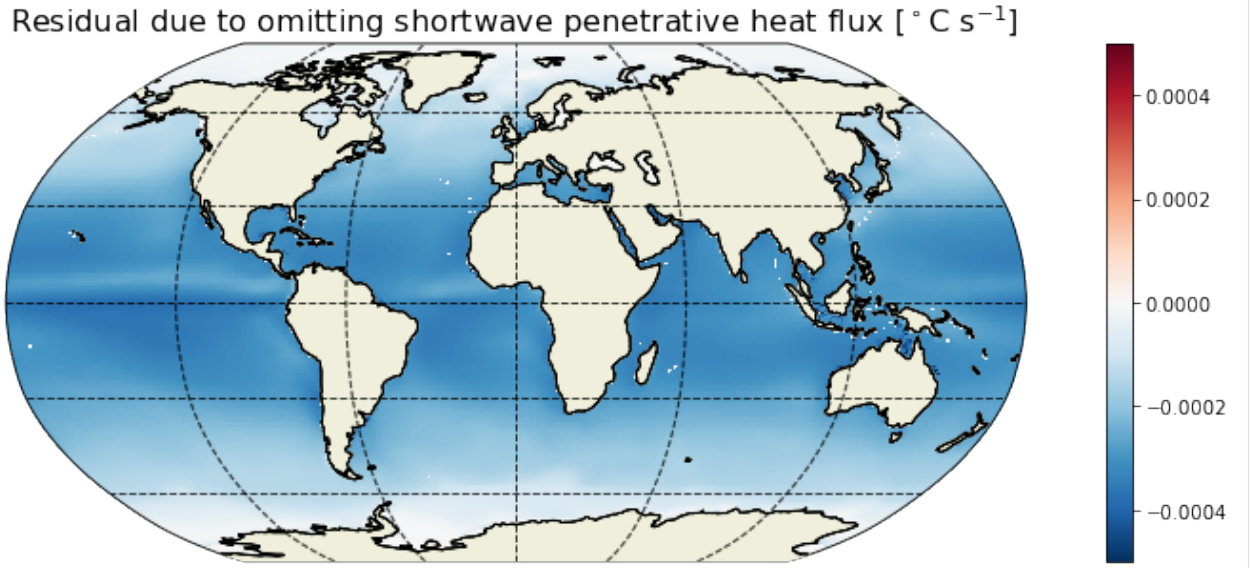


We see that the contribution from geothermal flux in the heat budget is well above the residual (by *three orders of*

magnitude).

[71]:
```
# Residual when omitting shortwave penetrative heat flux
res_sw = (rhs-SWpen-G_total).sum(dim='k').sum(dim='time').compute()
```

[72]:
```
plt.figure(figsize=(15,5))
ecco.plot_proj_to_latlon_grid(ecco_grid.XC, ecco_grid.YC, res_sw,
                              cmin=-5e-4, cmax=5e-4, show_colorbar=True, cmap='RdBu_r
↪', dx=0.2, dy=0.2)
plt.title(r'Residual due to omitting shortwave penetrative heat flux [$^\circ$C s$^{-
↪1}$] ', fontsize=16)
plt.show()
```



In terms of subsurface heat fluxes, shortwave penetration represents a much larger heat flux compared to geothermal heat flux (by around *three orders of magnitude*).

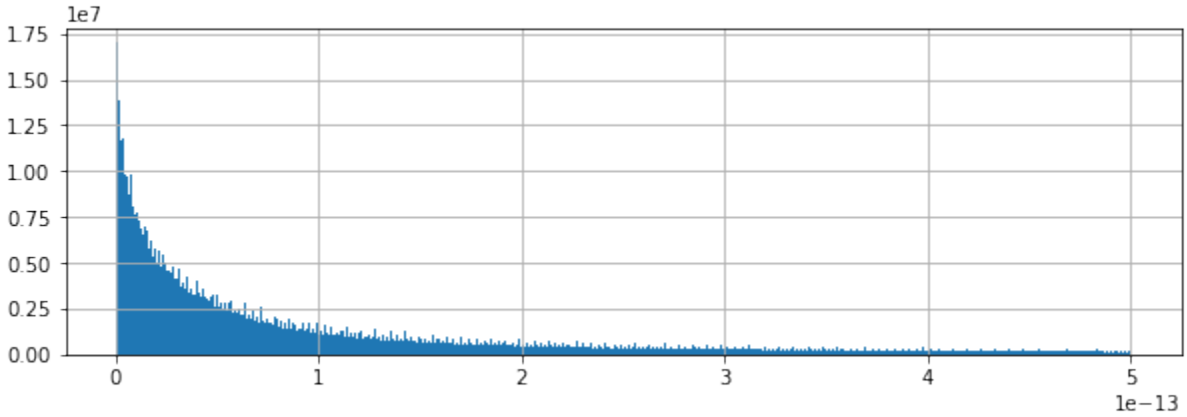
4.11.2 Histogram of residuals

We can look at the distribution of residuals to get a little more confidence.

[76]:
```
from xhistogram.xarray import histogram
```

[74]:
```
tmp = np.abs(rhs-G_total).values.ravel()
```

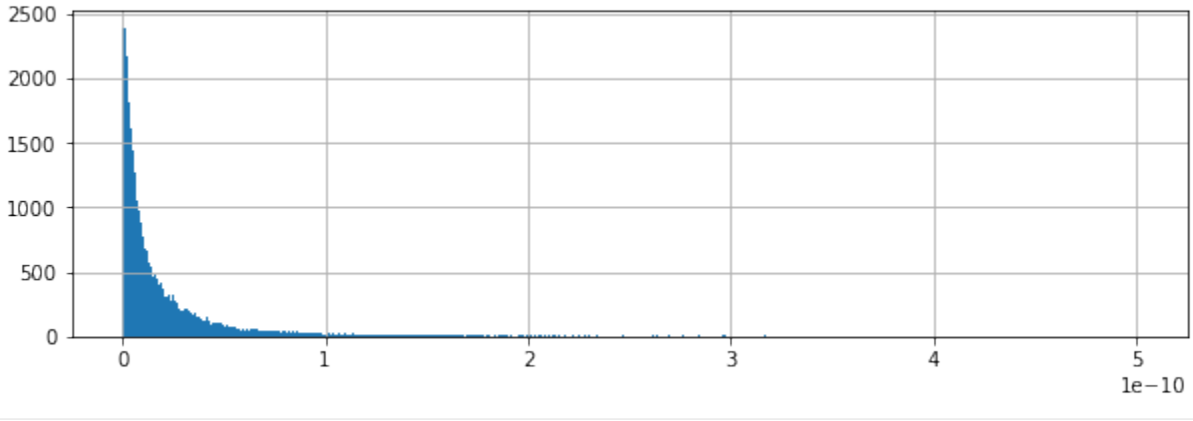
[87]:
```
plt.figure(figsize=(10,3));

plt.hist(tmp[np.nonzero(tmp > 0)],np.linspace(0, .5e-12,501));
plt.grid()
```



Almost all residuals < 10^{-13} °C s^{-1}.

[88]:
```
tmp = np.abs(res).values.ravel()
```

[89]:
```
plt.figure(figsize=(10,3));

plt.hist(tmp[np.nonzero(tmp > 0)],np.linspace(0, .5e-9, 1000));
plt.grid()
```



Summing residuals vertically and temporally yields < 10^{-10} °C s^{-1} for most grid points.

4.12 Heat budget closure through time

4.12.1 Global average budget closure

Another way of demonstrating heat budget closure is to show the global spatially-averaged `THETA` tendency terms

[91]:
```
# Volume (m^3)
vol = (ecco_grid.rA*ecco_grid.drF*ecco_grid.hFacC).transpose('tile','k','j','i')

# Take volume-weighted mean of these terms
tmp_a=(G_total*vol).sum(dim=('k','i','j','tile'))/vol.sum()
tmp_b=(G_advection*vol).sum(dim=('k','i','j','tile'))/vol.sum()
```

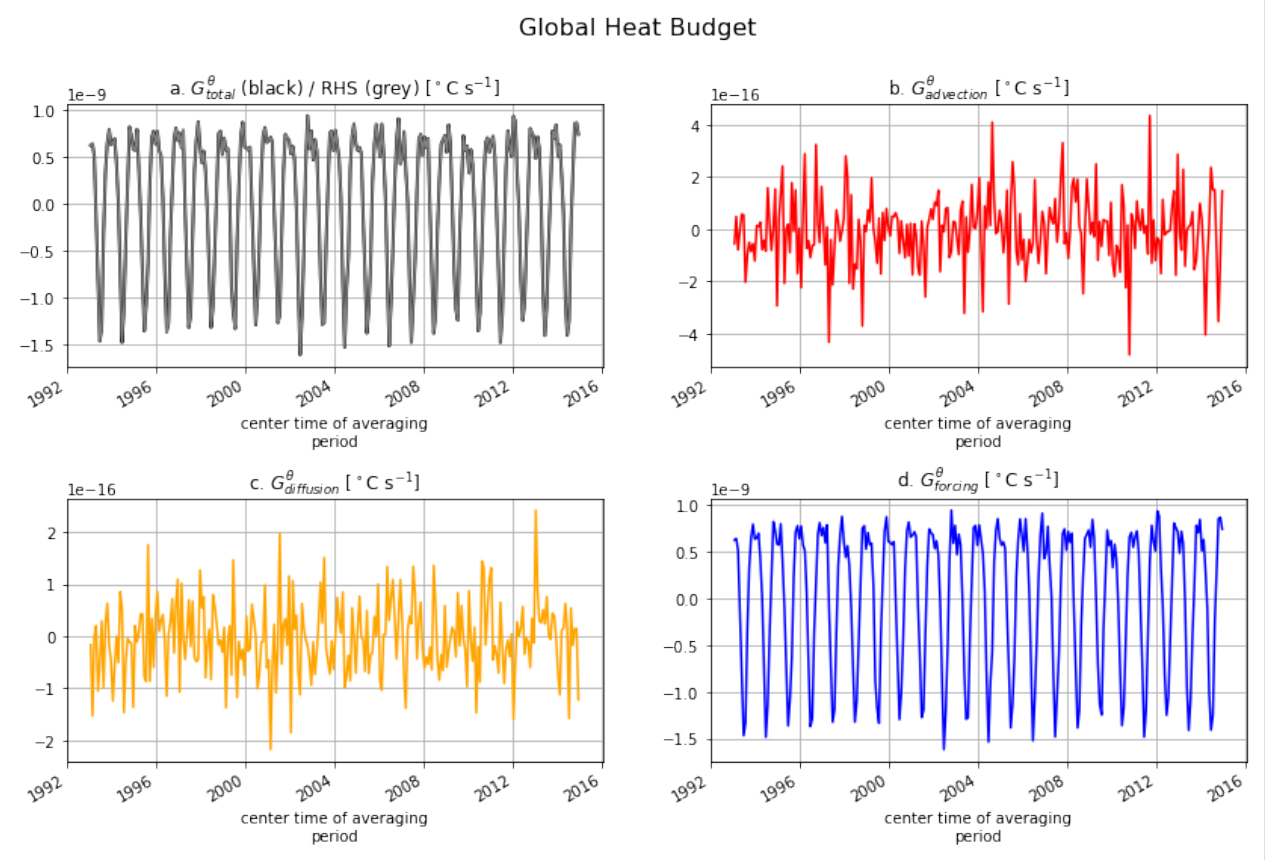
(continues on next page)

(continued from previous page)

```
tmp_c=(G_diffusion*vol).sum(dim=('k','i','j','tile'))/vol.sum()
tmp_d=(G_forcing*vol).sum(dim=('k','i','j','tile'))/vol.sum()
tmp_e=(rhs*vol).sum(dim=('k','i','j','tile'))/vol.sum()

# Result is five time series
tmp_a.dims
```

[91]:
```
('time',)
```

[92]:
```
fig, axs = plt.subplots(2, 2, figsize=(14,8))

plt.sca(axs[0,0])
tmp_a.plot(color='k',lw=2)
tmp_e.plot(color='grey')
axs[0,0].set_title(r'a. $G^\theta_{total}$ (black) / RHS (grey) [$^\circ$C s$^{-1}$]',
↪ fontsize=12)
plt.grid()

plt.sca(axs[0,1])
tmp_b.plot(color='r')
axs[0,1].set_title(r'b. $G^\theta_{advection}$ [$^\circ$C s$^{-1}$]', fontsize=12)
plt.grid()

plt.sca(axs[1,0])
tmp_c.plot(color='orange')
axs[1,0].set_title(r'c. $G^\theta_{diffusion}$ [$^\circ$C s$^{-1}$]', fontsize=12)
plt.grid()

plt.sca(axs[1,1])
tmp_d.plot(color='b')
axs[1,1].set_title(r'd. $G^\theta_{forcing}$ [$^\circ$C s$^{-1}$]', fontsize=12)
plt.grid()
plt.subplots_adjust(hspace = .5, wspace=.2)
plt.suptitle('Global Heat Budget', fontsize=16);
```



When averaged over the entire ocean the ocean heat transport terms ($G^{\theta}_{\text{advection}}$ and $G^{\theta}_{\text{diffusion}}$) have no net impact on $G^{\theta}_{\text{total}}$ (i.e., $\partial\theta/\partial t$). This makes sence because $G^{\theta}_{\text{advection}}$ and $G^{\theta}_{\text{diffusion}}$ can only redistributes heat. Globally, θ can only change via $G^{\theta}_{\text{forcing}}$.

4.12.2 Local heat budget closure

Locally we expect that heat divergence can impact θ. This is demonstrated for a single grid point.

[93]:
```
# Pick any set of indices (tile, k, j, i) corresponding to an ocean grid point
t,k,j,i = (6,10,40,29)
print(t,k,j,i)
```

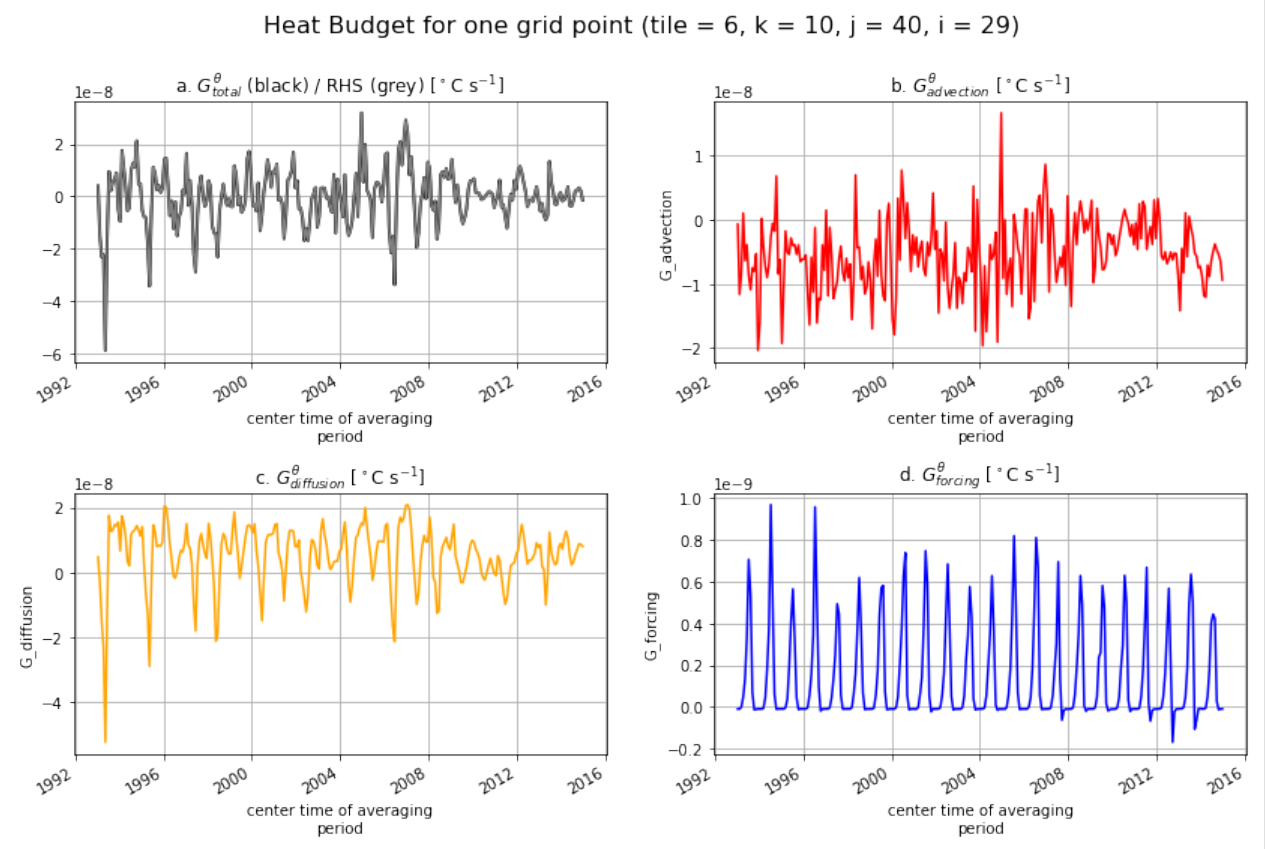
```
6 10 40 29
```

[94]:
```
tmp_a = G_total.isel(tile=t,k=k,j=j,i=i)
tmp_b = G_advection.isel(tile=t,k=k,j=j,i=i)
tmp_c = G_diffusion.isel(tile=t,k=k,j=j,i=i)
tmp_d = G_forcing.isel(tile=t,k=k,j=j,i=i)
tmp_e = rhs.isel(tile=t,k=k,j=j,i=i)

fig, axs = plt.subplots(2, 2, figsize=(14,8))

plt.sca(axs[0,0])
tmp_a.plot(color='k',lw=2)
tmp_e.plot(color='grey')
axs[0,0].set_title(r'a. $G^\theta_{total}$ (black) / RHS (grey) [$^\circ$C s$^{-1}$]',
↪ fontsize=12)
```

(continues on next page)

(continued from previous page)

```
plt.grid()

plt.sca(axs[0,1])
tmp_b.plot(color='r')
axs[0,1].set_title(r'b. $G^\theta_{advection}$ [$^\circ$C s$^{-1}$]', fontsize=12)
plt.grid()

plt.sca(axs[1,0])
tmp_c.plot(color='orange')
axs[1,0].set_title(r'c. $G^\theta_{diffusion}$ [$^\circ$C s$^{-1}$]', fontsize=12)
plt.grid()

plt.sca(axs[1,1])
tmp_d.plot(color='b')
axs[1,1].set_title(r'd. $G^\theta_{forcing}$ [$^\circ$C s$^{-1}$]', fontsize=12)
plt.grid()
plt.subplots_adjust(hspace = .5, wspace=.2)
plt.suptitle('Heat Budget for one grid point (tile = %i, k = %i, j = %i, i = %i)'%(t,
↪k,j,i), fontsize=16);
```



Indeed, the heat divergence terms do contribute to θ variations at a single point. Local heat budget closure is also confirmed at this grid point as we see that the sum of terms on the RHS (grey line) equals the LHS (black line).

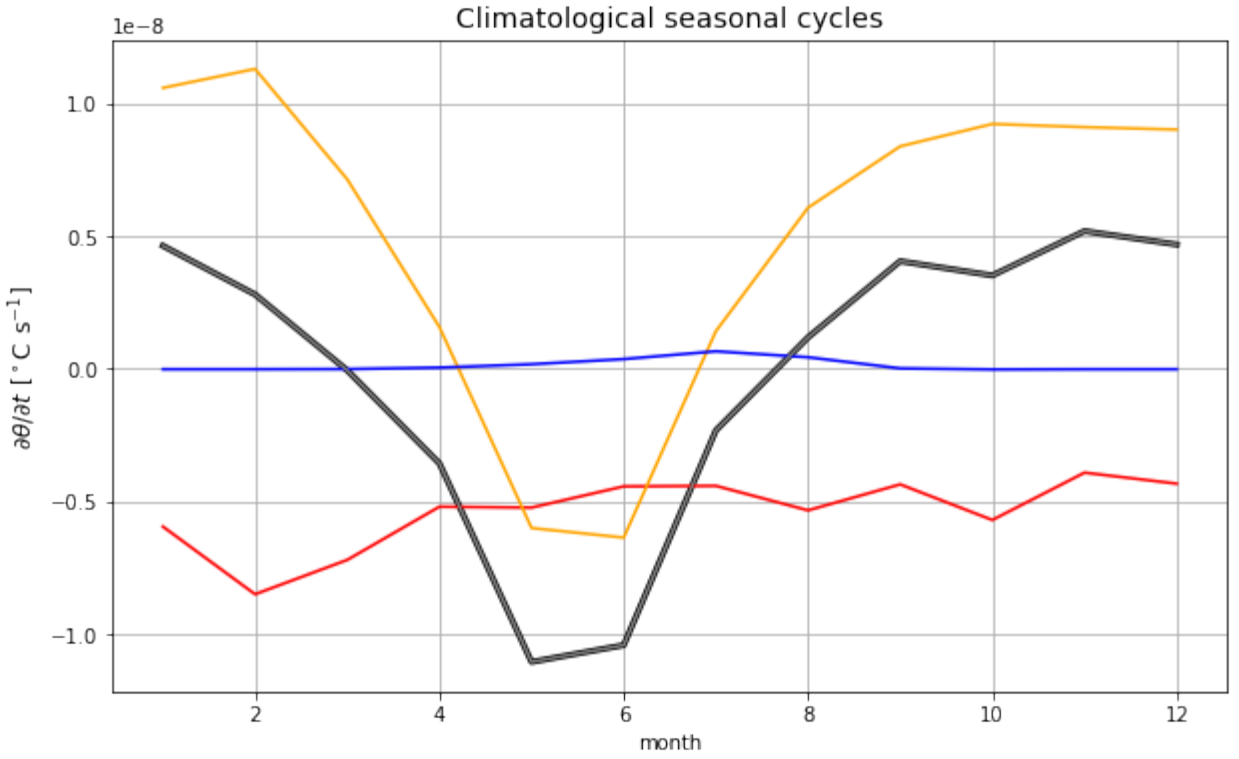
For the Arctic grid point, there is a clear seasonal cycles in both G^θ_{total}, $G^\theta_{\text{diffusion}}$ and $G^\theta_{\text{forcing}}$. The seasonal cycle in $G^\theta_{\text{forcing}}$ seems to be the reverse of G^θ_{total} and $G^\theta_{\text{diffusion}}$.

[95]:
```
plt.figure(figsize=(10,6));
```

(continues on next page)

(continued from previous page)

```
tmp_a.groupby('time.month').mean('time').plot(color='k',lw=3)
tmp_b.groupby('time.month').mean('time').plot(color='r')
tmp_c.groupby('time.month').mean('time').plot(color='orange')
tmp_d.groupby('time.month').mean('time').plot(color='b')
tmp_e.groupby('time.month').mean('time').plot(color='grey')
plt.ylabel(r'$\partial\theta$/$\partial t$ [$^\circ$C s$^{-1}$]', fontsize=12)
plt.grid()
plt.title('Climatological seasonal cycles', fontsize=14)
plt.show()
```



The mean seasonal cycle of the total is driven by seasonality in diffusion. However, this is likely depth-dependent. How does the balance look across the upper 200 meter at that location?

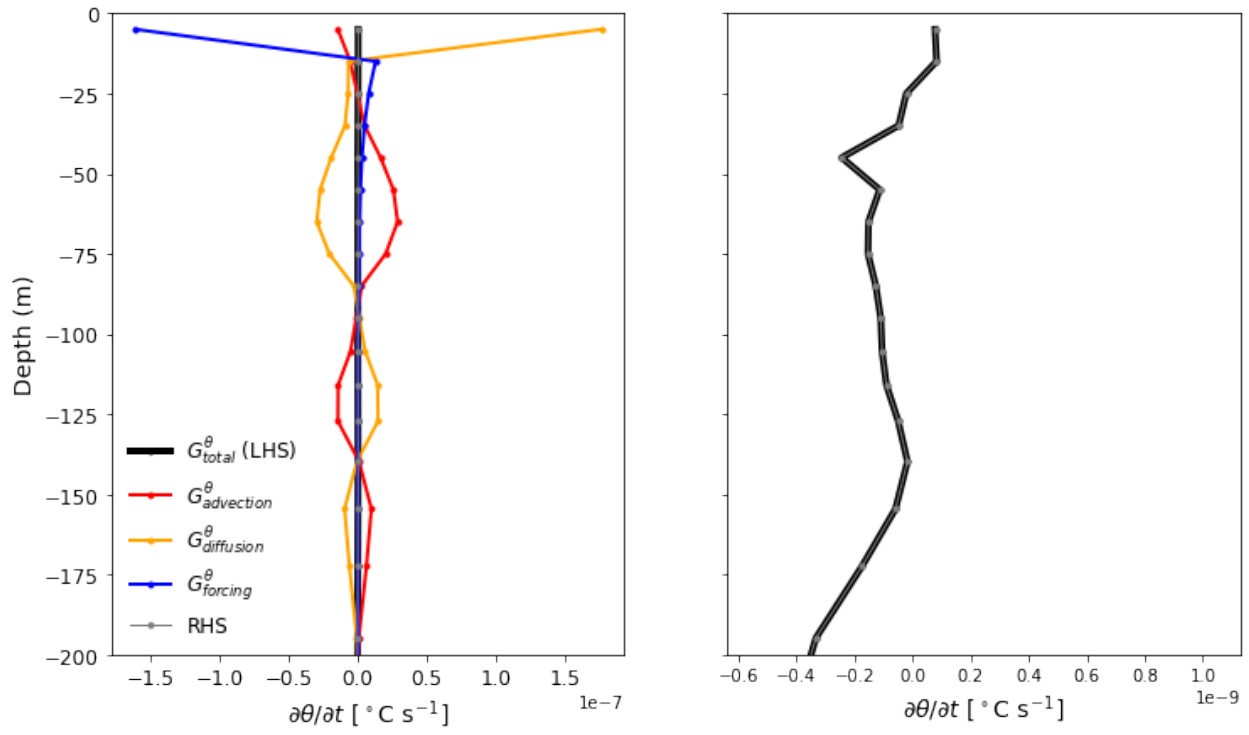
4.13 Time-mean vertical profiles

[97]:
```
fig = plt.subplots(1, 2, sharey=True, figsize=(12,7))

plt.subplot(1, 2, 1)
plt.plot(G_total.isel(tile=t,j=j,i=i).mean('time'), ecco_grid.Z,
         lw=4, color='black', marker='.', label=r'$G^\theta_{total}$ (LHS)')

plt.plot(G_advection.isel(tile=t,j=j,i=i).mean('time'), ecco_grid.Z,
         lw=2, color='red', marker='.', label=r'$G^\theta_{advection}$')

plt.plot(G_diffusion.isel(tile=t,j=j,i=i).mean('time'), ecco_grid.Z,
         lw=2, color='orange', marker='.', label=r'$G^\theta_{diffusion}$')
```

(continues on next page)

(continued from previous page)

```
plt.plot(G_forcing.isel(tile=t,j=j,i=i).mean('time'), ecco_grid.Z,
         lw=2, color='blue', marker='.', label=r'$G^\theta_{forcing}$')
plt.plot(rhs.isel(tile=t,j=j,i=i).mean('time'), ecco_grid.Z, lw=1, color='grey',␣
↪marker='.', label='RHS')
plt.xlabel(r'$\partial\theta$/$\partial t$ [$^\circ$C s$^{-1}$]', fontsize=14)
plt.ylim([-200,0])
plt.ylabel('Depth (m)', fontsize=14)
plt.gca().tick_params(axis='both', which='major', labelsize=12)
plt.legend(loc='lower left', frameon=False, fontsize=12)

plt.subplot(1, 2, 2)
plt.plot(G_total.isel(tile=t,j=j,i=i).mean('time'), ecco_grid.Z,
         lw=4, color='black', marker='.', label=r'$G^\theta_{total}$ (LHS)')
plt.plot(rhs.isel(tile=t,j=j,i=i).mean('time'), ecco_grid.Z, lw=1, color='grey',␣
↪marker='.', label='RHS')
plt.setp(plt.gca(), 'yticklabels',[])
plt.xlabel(r'$\partial\theta$/$\partial t$ [$^\circ$C s$^{-1}$]', fontsize=14)
plt.ylim([-200,0])
plt.show()
```



Balance between surface forcing and diffusion in the top layers. Balance between advection and diffusion at depth.

[ ]:

CHAPTER 5

Indices and tables

- genindex
- modindex
- search