[1]:

```python
# Description: Plot the time-mean cross-isobath heat transport (mean and
#              and display Table S1 (supplementary materials).
#
#              -Raw mean/eddy HTs along 1000 m isobath.
#              -Cumulative sums of mean/eddy HTs.
#              -Cumulative sums of top, middle and bottom HTs
#              -Segment-integrated onshore HTs
#              -Along-shelf HT convergences
#              -Segment-integrated onshore volume transport.
#
#
# Author:      André Palóczy
# E-mail:      paloczy@gmail.com
# Date:        December/2017

import sys
sys.path.append('../../misc')
sys.path.append('/home/andre/Dropbox/python-modules/pygamman')

import warnings
warnings.filterwarnings('ignore')

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon
from netCDF4 import Dataset, num2date
from datetime import datetime
from cmocean import cm as cmo
from gsw import distance
from pandas import Series, DataFrame, concat
from local_utils import crosscorr, Tdecorr, rsig, angle_isobath
from local_utils import lon360to180, blksum, blkavg, blkavgt, near, near2


def _get_segmsk(x, y, d, seglims, dateline=False):
    xl, xr, yl, yr = seglims
    idxl = near2(x, y, xl, yl, return_index=True)[0]
    idxr = near2(x, y, xr, yr, return_index=True)[0]

    if d[idxl] > d[idxr]:
        idxl, idxr = idxr, idxl

    fseg = np.zeros(x.size)
    #print(idxl,idxr)
    if dateline:
        fseg[:idxl] = 1
        fseg[idxr:] = 1
    else:
        fseg[idxl:idxr] = 1
```

```python
        return np.bool8(fseg)


    plt.rc('text.latex', preamble=r'\usepackage{amsmath}')
    ##---
    plt.close('all')

    GET_HT_DIVERGENCE = True
    REVERSED_XAXIS = True
    isob = 1000
    TW2GW = 1e3
    # t_afterspinup = datetime(1969, 1, 1) # Cap first 10 years (spin-up).
    t_afterspinup = datetime(1959, 1, 1)
    # t_afterspinup = datetime(2005, 1, 1)

    # Get segment lat/lon limits.
    segs_lims = {
    'Ross':[165., -150., -79., -68.],
    'Byrd':[-150., -130., -79., -68.],
    'Amundsen':[-130, -100, -76., -64.],
    'Bellingshausen':[-100., -75., -77., -60.],
    'S-AP':[-75., -64., -74., -60.],
    'N-AP':[-64., -53., -74., -60.],
    'Weddell':[-53., -11., -78., -59.],
    'W-EA':[-11., 65., -72., -60.],
    'C-EA':[65., 100., -72., -60.],
    'E-EA':[100., 165., -72., -60.]
    }
    segnames = segs_lims.keys()

    # Load time-varying heat budget file.
    head = '../../data_reproduce_figs/'
    fnamez = 'hflxmelt_tseries%dm.npz'%isob

    fnamez = head + 'large_files/' + fnamez

    try:
        dd = np.load(fnamez, encoding='bytes')
    except FileNotFoundError:
        print("*ERROR*: File %s not found. Make sure you have downloaded it a
        sys.exit(1)

    for k in dd.keys():
        vars().update({k:dd[k]})

    # Get model topography.
    ftopo = head + 'POP_topog.nc'
    nctopo = Dataset(ftopo).variables
    fcap = 501
    cm2m = 1e-2
    lont, latt, ht = nctopo['TLONG'][:fcap,:], nctopo['TLAT'][:fcap,:], nctop
    lont = lon360to180(lont)
```

```
# Take only part of the simulation after spin-up.
f_afterspinup = t>=t_afterspinup
UQx = UQx[f_afterspinup,:]
UQxm = UQxm[f_afterspinup,:]
UQxe = UQxe[f_afterspinup,:]
Ux0 = Ux.copy()
Ux = Ux[f_afterspinup,:]
UQxm_100m = UQxm_100m[f_afterspinup,:]
UQxe_100m = UQxe_100m[f_afterspinup,:]
UQxm_100m_700m = UQxm_100m_700m[f_afterspinup,:]
UQxe_100m_700m = UQxe_100m_700m[f_afterspinup,:]
UQxm_700m_1000m = UQxm_700m_1000m[f_afterspinup,:]
UQxe_700m_1000m = UQxe_700m_1000m[f_afterspinup,:]

fnamez_isobs = head + 'isobaths.nc'
ncxm = Dataset(fnamez_isobs)
Im = ncxm["%d m isobath (U-points)"%isob]['i'][:]
Jm = ncxm["%d m isobath (U-points)"%isob]['j'][:]

# Calculate time-averaged mean and eddy components.
# Plot their mean values and their associated variances,
# i.e., standard deviation (in time) for each point (along the isobath).
uqx = UQx.mean(axis=0)
uqxm = UQxm.mean(axis=0)
uqxe = UQxe.mean(axis=0)
uqx_tstd = UQx.std(axis=0)
uqxm_tstd = UQxm.std(axis=0)
uqxe_tstd = UQxe.std(axis=0)
#
uqxmsrf = UQxm_100m.mean(axis=0)
uqxesrf = UQxe_100m.mean(axis=0)
uqxmmid = UQxm_100m_700m.mean(axis=0)
uqxemid = UQxe_100m_700m.mean(axis=0)
uqxmbot = UQxm_700m_1000m.mean(axis=0)
uqxebot = UQxe_700m_1000m.mean(axis=0)

# Time series of total heat transports within each layer.
uqxsrf = uqxmsrf + uqxesrf # 0 - 100 m.
uqxmid = uqxmmid + uqxemid # 100 - 700 m.
uqxbot = uqxmbot + uqxebot # 700 - 1000 m.

# Use longitude as independent coordinate instead of along-isobath distan
dist = d.copy() # [km].
d = x.copy()

# Volume transport.
ux = Series(Ux.mean(axis=0), index=dist*1e3)
ux = ux.rolling(window=19, win_type='hanning', center=True).mean() # smoo

if REVERSED_XAXIS:
    x = np.flipud(x)
```

```
    y = np.flipud(y)
    d = np.flipud(d) # Recall that d is now longitude, so ok to reverse.
    uqxm = np.flipud(uqxm)
    uqxe = np.flipud(uqxe)
    uqx = np.flipud(uqx)
    uqxsrf = np.flipud(uqxsrf)
    uqxmid = np.flipud(uqxmid)
    uqxbot = np.flipud(uqxbot)

# Block-average (in space) the standard deviations.
Lx = 30 # [km], Desired size for each block.
dx = 3.28971284 # [km].
blkstd = int(Lx/dx)
davg, uqxm_tstdavgx, _ = blkavg(d, uqxm_tstd, every=blkstd)
_, uqxe_tstdavgx, _ = blkavg(d, uqxe_tstd, every=blkstd)

# Calculate segment-averaged heat transports.
uqxmseg_avg = dict()
uqxeseg_avg = dict()
uqxseg_avg = dict()
uqxmseg_std = dict()
uqxeseg_std = dict()
uqxseg_std = dict()
uqxseg_total = dict()
uxseg_total = dict()

didx = np.load(head+'segment_bdry_indices.npz')
idxls, idxrs = didx['idxl'].tolist(), didx['idxr'].tolist()

if REVERSED_XAXIS:
    x2, y2 = np.flipud(x), np.flipud(y)
else:
    x2, y2 = x.copy(), y.copy()

uqxmseg_totaltseries = dict()
uqxeseg_totaltseries = dict()
uqxseg_totaltseries = dict()
uxseg_totaltseries = dict()
# fig, ax = plt.subplots()
for seg in segs_lims.keys():
    idxl = idxls[seg]
    idxr = idxrs[seg]
    fseg = np.int32(np.zeros(x.size))
    if seg=='Ross':
        fseg[:idxl] = 1
        fseg[idxr:] = 1
        aux = dist[fseg]
        Lxseg = dist[idxl] - dist[0] + dist[-1] - dist[idxr] # 180 to 165
    else:
        fseg[idxl:idxr] = 1
        aux = dist[fseg]
        Lxseg = dist[idxr] - dist[idxl]
```

```
        fseg = np.bool8(fseg)

        if REVERSED_XAXIS:
            fseg = np.flipud(fseg)

        print(seg, int(Lxseg),'km, ','%d-->%d'%(idxl,idxr))
        #ax.plot(x[fseg], y[fseg], marker='.')

        # Time averages of the heat transports averaged along each segment.
        uqxmseg = uqxm[fseg].sum()*TW2GW/Lxseg # Mean, [GW/km].
        uqxeseg = uqxe[fseg].sum()*TW2GW/Lxseg # Eddy, [GW/km].
        uqxseg_tot = uqx[fseg].sum()
        uxseg0 = Ux0[:, fseg].sum(axis=1)
        uxseg = Ux[:, fseg].sum(axis=1)
        uqxmseg_totaltseries.update({seg:UQxm[:, fseg].sum(axis=1)*TW2GW/Lxse
        uqxeseg_totaltseries.update({seg:UQxe[:, fseg].sum(axis=1)*TW2GW/Lxse
        uqxseg_totaltseries.update({seg:UQx[:, fseg].sum(axis=1)*TW2GW/Lxseg}
        uxseg_totaltseries.update({seg:uxseg})
        uqxseg = uqxseg_tot*TW2GW/Lxseg        # Total, [GW/km].
        uqxseg_total.update({seg:uqxseg_tot})  # [TW].
        uxseg_total.update({seg:uxseg0})       # [Sv].

        # Standard deviations of the time-averaged heat transports averaged a
        uqxmsegstd = uqxm_tstd[fseg].sum()*TW2GW/Lxseg
        uqxesegstd = uqxe_tstd[fseg].sum()*TW2GW/Lxseg
        uqxsegstd = uqx_tstd[fseg].sum()*TW2GW/Lxseg

        uqxmseg_avg.update({seg:uqxmseg})
        uqxeseg_avg.update({seg:uqxeseg})
        uqxseg_avg.update({seg:uqxseg})
        uqxmseg_std.update({seg:uqxmsegstd})
        uqxeseg_std.update({seg:uqxesegstd})
        uqxseg_std.update({seg:uqxsegstd})

    # Integral timescales of Ux, UQxm and UQxe (in months).
    nblk = 51 # Each block is 1 year long.
    # _ = [print(max(Tdecorr(crosscorr(uxseg_totaltseries[seg],uxseg_totaltse
    segnames = list(segs_lims.keys())

    # Put it in a pandas DataFrame and print in latex.
    col1, col2, col3, col4 = dict(), dict(), dict(), dict()
    for seg in segnames:
        a = uxseg_totaltseries[seg] # On-shelf volume transport.
        col1.update({seg:Tdecorr(crosscorr(a, a, nblk)[1]).max()})
        a = uqxmseg_totaltseries[seg] # On-shelf mean heat transport.
        col2.update({seg:Tdecorr(crosscorr(a, a, nblk)[1]).max()})
        a = uqxeseg_totaltseries[seg] # On-shelf eddy heat transport.
        col3.update({seg:Tdecorr(crosscorr(a, a, nblk)[1]).max()})
        a = uqxseg_totaltseries[seg] # On-shelf total heat transport.
        col4.update({seg:Tdecorr(crosscorr(a, a, nblk)[1]).max()})

    col1tab1 = DataFrame.from_dict(col1, orient='index', columns=[r'U'])
```

```
col2tab1 = DataFrame.from_dict(col2, orient='index', columns=[r'UQxm'])
col3tab1 = DataFrame.from_dict(col3, orient='index', columns=[r'UQxe'])
col4tab1 = DataFrame.from_dict(col4, orient='index', columns=[r'UQx'])
table1 = concat((col1tab1,col2tab1,col3tab1,col4tab1), axis=1)
table1_latex = table1.to_latex(float_format='%1.2f').replace('\\\\','\\')

# Maximum absolute value of correlation and lag at which it happens.
col1, col2, col3, col4 = dict(), dict(), dict(), dict()
for seg in segnames:
    a = uxseg_totaltseries[seg] # On-shelf volume transport.
    b = uqxmseg_totaltseries[seg] # On-shelf mean heat transport.
    c = crosscorr(a, b, nblk)[1]
    idm = np.nanargmax(np.abs(c))
    col1.update({seg:(c[idm],idm)})
    b = uqxeseg_totaltseries[seg] # On-shelf eddy heat transport.
    c = crosscorr(a, b, nblk)[1]
    idm = np.nanargmax(np.abs(c))
    col2.update({seg:(c[idm],idm)})
    b = uqxseg_totaltseries[seg] # On-shelf total heat transport.
    c = crosscorr(a, b, nblk)[1]
    idm = np.nanargmax(np.abs(c))
    col3.update({seg:(c[idm],idm)})
    # MEan-eddy heat transport correlation.
    a = uqxmseg_totaltseries[seg]
    b = uqxeseg_totaltseries[seg]
    c = crosscorr(a, b, nblk)[1]
    idm = np.nanargmax(np.abs(c))
    col4.update({seg:(c[idm],idm)})

col1tab2 = DataFrame.from_dict(col1, orient='index')
col2tab2 = DataFrame.from_dict(col2, orient='index')
col3tab2 = DataFrame.from_dict(col3, orient='index')
col4tab2 = DataFrame.from_dict(col4, orient='index')
table_S1 = concat((col1tab2,col2tab2,col3tab2,col4tab2), axis=1)
table_S1_latex = table_S1.to_latex(float_format='%1.3f').replace('\\\\','

# Numer of Effective DoF for each cross-correlation and associated minimu
# Take the integral timescale of the slower of the two variables for each
nseries = 612/nblk # Number of months on each independent time series.
edof_UUQxm = nseries/concat((col1tab1, col2tab1), axis=1).max(axis='colum
edof_UUQxe = nseries/concat((col1tab1, col3tab1), axis=1).max(axis='colum
edof_UUQx = nseries/concat((col1tab1, col4tab1), axis=1).max(axis='column
edof_UQxmUQxe = nseries/concat((col2tab1, col3tab1), axis=1).max(axis='co
alpha = 0.95
print('')
print("=== U-UQxm ===")
_ = [print('Minimum significant r for %1.3f EDoF at %d %%: %1.3f'%(edof,
print("=== U-UQxe ===")
_ = [print('Minimum significant r for %1.3f EDoF at %d %%: %1.3f'%(edof,
print("=== U-UQxt ===")
_ = [print('Minimum significant r for %1.3f EDoF at %d %%: %1.3f'%(edof,
print("=== UQxm-UQxe ===")
```

```python
_ = [print('Minimum significant r for %1.3f EDoF at %d %%: %1.3f'%(edof,

df1 = np.abs(col1tab2[0])>=rsig(edof_UUQxm, alpha=alpha)
df2 = np.abs(col2tab2[0])>=rsig(edof_UUQxe, alpha=alpha)
df3 = np.abs(col3tab2[0])>=rsig(edof_UUQx, alpha=alpha)
df4 = np.abs(col4tab2[0])>=rsig(edof_UQxmUQxe, alpha=alpha)
issig_table_S1 = concat((df1, df2, df3, df4), axis=1)

# lon, lat, ht must start at -180 to plot contour.
fcut = np.where(np.abs(np.diff(lont[0,:]))>10)[0][0] + 1
lont = np.hstack((lont[:,fcut:], lont[:,:fcut]))
latt = np.hstack((latt[:,fcut:], latt[:,:fcut]))
ht = np.hstack((ht[:,fcut:], ht[:,:fcut]))
```

```
Ross 2836 km,  540-->5808
Byrd 933 km,  540-->811
Amundsen 2022 km,  811-->1338
Bellingshausen 1413 km,  1338-->1686
S-AP 1074 km,  1686-->1915
N-AP 888 km,  1915-->2083
Weddell 4339 km,  2083-->3124
W-EA 4988 km,  3124-->4286
C-EA 2393 km,  4286-->4805
E-EA 4626 km,  4805-->5808

=== U-UQxm ===
Minimum significant r for 3.203 EDoF at 95 %: 1.095
Minimum significant r for 9.323 EDoF at 95 %: 0.642
Minimum significant r for 9.893 EDoF at 95 %: 0.623
Minimum significant r for 13.080 EDoF at 95 %: 0.542
Minimum significant r for 26.034 EDoF at 95 %: 0.384
Minimum significant r for 18.628 EDoF at 95 %: 0.454
Minimum significant r for 10.906 EDoF at 95 %: 0.593
Minimum significant r for 12.795 EDoF at 95 %: 0.548
Minimum significant r for 16.595 EDoF at 95 %: 0.481
Minimum significant r for 16.468 EDoF at 95 %: 0.483
=== U-UQxe ===
Minimum significant r for 3.203 EDoF at 95 %: 1.095
Minimum significant r for 9.323 EDoF at 95 %: 0.642
Minimum significant r for 9.893 EDoF at 95 %: 0.623
Minimum significant r for 13.080 EDoF at 95 %: 0.542
Minimum significant r for 26.466 EDoF at 95 %: 0.381
Minimum significant r for 16.655 EDoF at 95 %: 0.480
Minimum significant r for 10.906 EDoF at 95 %: 0.593
Minimum significant r for 15.006 EDoF at 95 %: 0.506
Minimum significant r for 16.595 EDoF at 95 %: 0.481
Minimum significant r for 16.468 EDoF at 95 %: 0.483
=== U-UQxt ===
```

[2]

```python
# Display Table S1 (supplemental materials).
table_S1.head(n=10)
```

	0	1	0	1	0	1	0

	0	1	0	1	0	1	0
Ross	0.906166	10	-0.981256	10	-0.397188	1	-0.871
Byrd	0.587098	5	-0.926215	10	0.786725	5	-0.627
Amundsen	0.502324	17	-0.993026	10	0.639297	18	-0.504
Bellingshausen	0.658527	10	-0.991616	10	-0.450980	20	-0.640
S-AP	0.959000	10	-0.992923	10	-0.896924	10	-0.963
N-AP	-0.702749	1	-0.998862	10	-0.909771	10	0.708
Weddell	0.872137	10	-0.990027	10	0.381370	11	-0.864
W-EA	0.720281	10	-0.973511	10	0.445624	7	-0.712
C-EA	0.558037	10	-0.990191	10	0.117829	1	-0.525
E-EA	0.451837	10	-0.989357	10	-0.450980	20	-0.420

[3]
```
# Display array with 'True' where the corresponding entry in Table S1 (su
# is statistically significant at the 95 % confidence level.
issig_table_S1.head(n=10)
```

	0	1	2	3
Ross	False	False	False	False
Byrd	False	True	True	True
Amundsen	False	True	True	False
Bellingshausen	True	True	False	True
S-AP	True	True	True	True
N-AP	True	True	True	True
Weddell	True	True	False	True
W-EA	True	True	False	True
C-EA	True	True	False	True
E-EA	False	True	False	False

[4]
```
# Estimate segment-averaged transports from Stewart et al. (2018)'s resul
stewart_etal2018_transports_MWpermeter_at1000m_isob = {'EA':2, 'Amery':2.
stewart_etal2018_Lsegs = {'EA':2690, 'Amery':1556, 'Maud':1986, 'Ross':23

phi_total_segs_stewart_etal2018 = {}
```

```python
for seg in stewart_etal2018_Lsegs.keys():
    Phi_Lavg = stewart_etal2018_transports_MWpermeter_at1000m_isob[seg]
    L = stewart_etal2018_Lsegs[seg]
    Phi = Phi_Lavg*L*1e-3 # [TW].
    phi_total_segs_stewart_etal2018.update({seg:Phi})
print(phi_total_segs_stewart_etal2018)

# In [30]: phi_total_segs_stewart_etal2018
# Out[30]:
# {'Amery': 3.89,                  ~C-EA
#  'Amundsen': 3.452,              ~Amundsen + Byrd (1.6 TW)
#  'Bellingshausen': 1.719,        ~Bellingshausen
#  'EA': 5.38,                     ~E-EA
#  'Maud': 0.19860000000000003,    ~W-EA
#  'Ross': 11.73,                  ~Ross + a piece of E-EA
#  'WAP': -2.4984,                 ~S-AP + N-AP
#  'Weddell': 2.7920000000000003} ~Weddell.

F = 0
for Fi in phi_total_segs_stewart_etal2018.values():
    F+=Fi

print('Net circumpolar transport (Stewart et al. (2018): %.2f TW'%F)
# Stewart et al. (2018)'s Sep/2011-Oct2012 average: 26.66 TW
# Dinniman et al. (2015)'s 2005-2009 average: ~50 TW
# This study's 2005-2009 average: ~20.3 TW
```

```
{'EA': 5.38, 'Amery': 3.89, 'Maud': 0.19860000000000003, 'Ross': 11.73,
'Amundsen': 3.452, 'Bellingshausen': 1.719, 'WAP': -2.4984, 'Weddell':
2.7920000000000003}
Net circumpolar transport (Stewart et al. (2018): 26.66 TW
```

[5]

```python
##################################################################
# Choose bounding longitudes for psimean and psieddy calculation.
##################################################################
segbrys_psi = {
'weddell_C':(-50.0, -25.0),   # Central Weddell
'Maud':(20, 60),              # Maud Coast, part of W-EA where there is th
'Amery_wE-EA':(55, 120),      # Combined Amery- and west part of E-EA. Sim
'E-EA_E':(122.7, 153.7),      # East E-EA peak UQxe area.
'Ross_W':(165.0, 180.0),      # W Ross, strong onshore UQxe and strong off
'E_Ross':(-173, -153),        # E Ross (~the gray stripe).
'Byrd_W':(-150, -135),        # West Byrd Coast (a localized cancellation)
'Amundsen_E':(-115.0, -102.34),  # E Amundsen (gray stripe).
'Bellingshausen_E':(-86.0, -75.0),     # E Bellingshausen (another gray s
}
npzname = head + 'segment_bdry_lonlatgrid_indices_for_streamfunction_calc
np.savez(npzname, segbrys_psi=segbrys_psi)
```

[6]

```python
############
```

```
# Figure 3 #
############

figsize = (10, 13) # (8, 10)
fig, ax = plt.subplots(nrows=6, figsize=figsize, sharex=True)

ax[2].plot(d, np.cumsum(uqxmsrf), 'm', linestyle='-', linewidth=0.8, alph
ax[2].plot(d, np.cumsum(uqxesrf), 'm', linestyle=':', linewidth=0.8, alph
ax[2].plot(d, np.cumsum(uqxmmid), 'k', linestyle='-', linewidth=0.8, alph
ax[2].plot(d, np.cumsum(uqxemid), 'k', linestyle=':', linewidth=0.8, alph
ax[2].plot(d, np.cumsum(uqxmbot), 'c', linestyle='-', linewidth=0.8, alph
ax[2].plot(d, np.cumsum(uqxebot), 'c', linestyle=':', linewidth=0.8, alph

ax[2].set_ylabel(r'$\int^y \text{HT}(y^\prime) \, dy^\prime$ [TW]', fonts

ax[1].plot(d, np.cumsum(uqxm), 'b', linewidth=0.80)
ax[1].plot(d, np.cumsum(uqxe), 'r', linewidth=0.80)
ax[1].plot(d, np.cumsum(uqx), 'k', linewidth=0.80)

ax[0].set_ylabel(r'HT [TW]', fontsize=16, fontweight='black', labelpad=2)

uqeddyfac = 1
# Block-average to smooth.
Lx = 30 # [km], Desired size for each block.
dx = 3.28971284 # [km].
blks = int(Lx/dx)
_, uqxm_sumx = blksum(d, uqxm, every=blks)
dsum, uqxe_sumx = blksum(d, uqxe, every=blks)
uqx_sumx = uqxm_sumx + uqxe_sumx
ax[0].plot(dsum, uqxm_sumx, 'b', linewidth=0.5)
ax[0].plot(dsum, uqxe_sumx*uqeddyfac, 'r', linewidth=0.5)

ax0dummy = ax[0].twiny()
ax0dummy.xaxis.set_tick_params(labelsize=9, labelbottom=False, labeltop=T
ax0dummy.grid(False)
ax0dummy.plot(dist, dist*0, linestyle='none')
ax0dummy.set_xlim((0, dist.max()))

ax[2].axhline(y=0, color='k', linewidth=1.0, linestyle='-', alpha=0.8)
ax[1].axhline(y=0, color='k', linewidth=1.0, linestyle='-', alpha=0.8)
ax[0].axhline(y=0, color='k', linewidth=1.0, linestyle='-', alpha=0.8)

# Mask deeper than 1000.
htdeep = 1000
fmsk = head + 'volmsk%dm.npz'%htdeep
mskdeep = ~np.bool8(np.load(fmsk)['volmsk'])
mskdeep = np.hstack((mskdeep[:,fcut:], mskdeep[:,:fcut]))
ht = np.ma.masked_where(mskdeep, ht)
ht[ht==0] = -1

# Plot model bottom topography.
dmin, dmax = 0, 1000
```

```
    dtk = 250
    cs = ax[5].pcolor(lont, latt, ht, cmap=cmo.deep, vmin=0, vmax=htdeep, zor
    # Mask land.
    cs.cmap.set_under(color=[0.5]*3)
    cs.cmap.set_over(alpha=0)
    cbaxes = fig.add_axes([0.53, 0.147, 0.30, 0.01])
    cb = plt.colorbar(mappable=cs, cax=cbaxes, orientation='horizontal')#, ex
    cb.set_ticks(np.arange(dmin, dmax+dtk, dtk))
    cb.ax.xaxis.set_tick_params(labelsize=8, pad=2)
    cb.ax.xaxis.set_ticks_position('top')
    cb.set_label(r'[m]', fontsize=9, fontweight='normal')

    ax[2].set_ylim(-70, 70)
    ax[2].set_yticks([-60, -45, -30, -15, 0, 15, 30, 45, 60])
    ax[1].set_yticks(np.arange(-25, 40, 5))
    ax[1].set_ylim(-30, 40)
    ax[0].set_ylim(-5, 5)
    ax[3].set_ylim(-5, 5)
    ax[4].set_ylim(-1.2, 1.2)
    ax[0].set_yticks(np.arange(-4, 6))
    ax[3].set_yticks([-4, -2, 0, 2, 4])
    ax[4].set_yticks([-1, -0.5, 0, 0.5, 1])
    ax[5].set_ylim(-80, -55)#-60)
    ax[2].xaxis.set_tick_params(labelsize=9)
    ax[2].yaxis.set_tick_params(labelsize=9)
    ax[1].yaxis.set_tick_params(labelsize=9)
    ax[0].xaxis.set_tick_params(labelsize=10)
    ax[0].yaxis.set_tick_params(labelsize=10)
    ax[3].yaxis.set_tick_params(labelsize=10)
    ax[4].yaxis.set_tick_params(labelsize=10)
    ax[5].xaxis.set_tick_params(labelsize=9)
    ax[5].yaxis.set_tick_params(labelsize=9)

    # Plot model 1000 m isobath.
    ax[5].plot(x, y, 'k', linewidth=0.8)

    dlontick = 30
    lon_ticks = np.arange(-180, 180+dlontick, dlontick)
    ax[5].axis('tight')
    ax[5].set_xticks(lon_ticks)
    ax0dummy.set_xlabel(r"Distance along 1000 m isobath [km]", fontsize=14, f
    ax[5].set_xlabel(r"Longitude", fontsize=16, fontweight='black')
    ax[5].set_ylabel(r"Latitude", fontsize=16, fontweight='black', labelpad=5

    # Vertical lines marking segments.
    kwvl = dict(color='gray', linestyle='--', linewidth=0.5)
    for seg in segs_lims.keys():
        lol, lor = segs_lims[seg][:2]
        ax[2].axvline(x=lol, **kwvl)
        ax[2].axvline(x=lor, **kwvl)
        ax[1].axvline(x=lol, **kwvl)
        ax[1].axvline(x=lor, **kwvl)
```

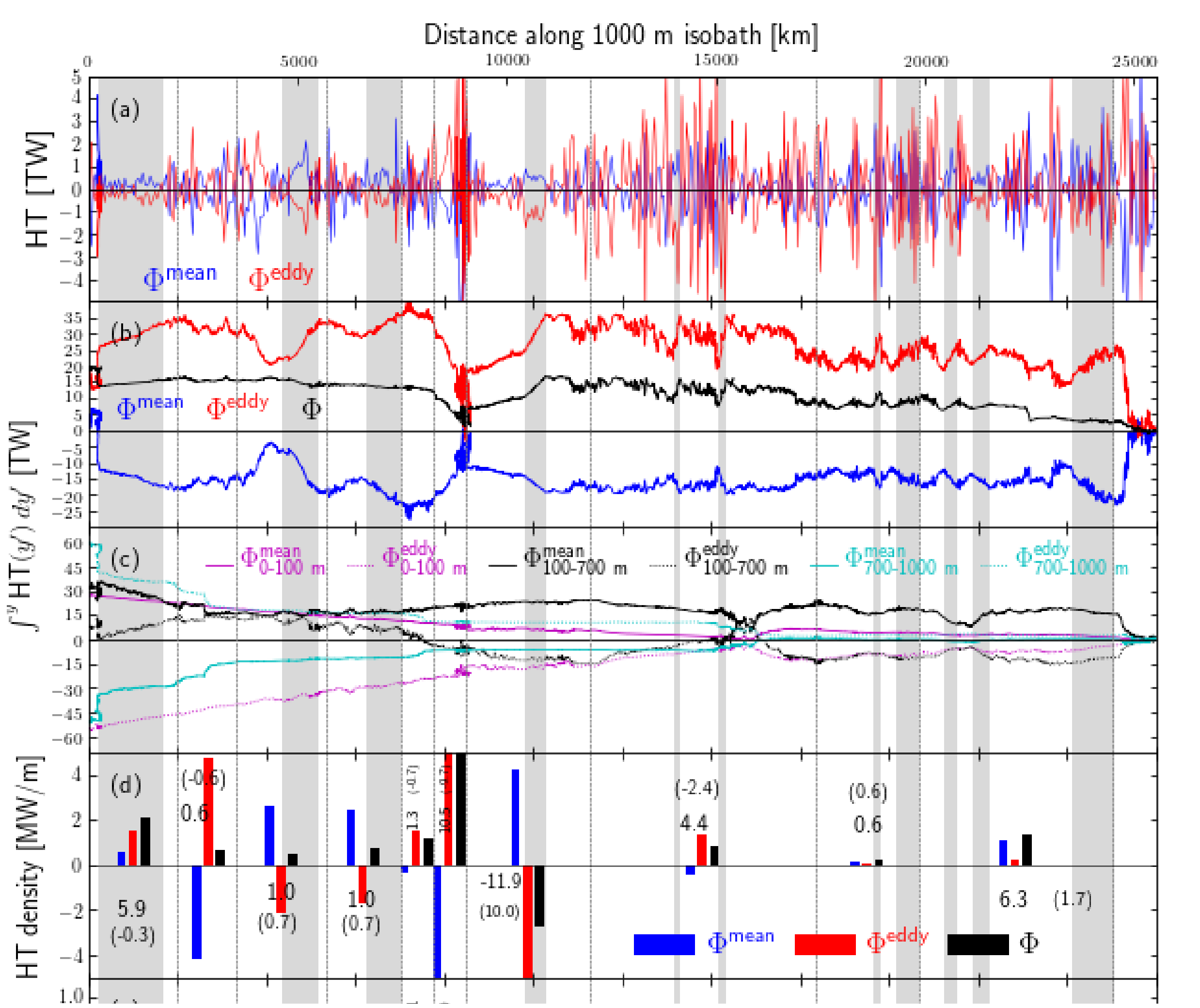
```
    ax[0].axvline(x=lol, **kwvl)
    ax[0].axvline(x=lor, **kwvl)
    ax[3].axvline(x=lol, **kwvl)
    ax[3].axvline(x=lor, **kwvl)
    ax[4].axvline(x=lol, **kwvl)
    ax[4].axvline(x=lor, **kwvl)
    ax[5].axvline(x=lol, **kwvl)
    ax[5].axvline(x=lor, **kwvl)

# Names of segments and longitudes to plot the names.
seg_txts = {r'E-EA':132.5, r'C-EA':82.5, r'Weddell':-32.0, r'Bellingshaus

# Plot segment-averaged heat transports.
dlo = 4
lonpltsc = dict()
lonpltsl = dict()
lonpltsr = dict()
uqxmseg_avgs = dict()
uqxeseg_avgs = dict()
uqxseg_avgs = dict()
for seg in segs_lims.keys():
    lonplti = seg_txts[seg]
    lonpltsc.update({seg:(lonplti)})
    lonpltsl.update({seg:(lonplti-dlo)})
    lonpltsr.update({seg:(lonplti+dlo)})
    uqxmseg_avgs.update({seg:uqxmseg_avg[seg]})
    uqxeseg_avgs.update({seg:uqxeseg_avg[seg]})
    uqxseg_avgs.update({seg:uqxseg_avg[seg]})

uqsegtxt = {'Ross':(-170, -2.0120672139699529),
'Byrd':(-149, 2.2711759699074783),
'Amundsen':(-119, -1.226805963592426),
'Bellingshausen':(-92.322580645161281, -1.5837428955822119),
'S-AP':(-70.258064516129025, 2.0),
'N-AP':(-59.225806451612897, 2.0),
'Weddell':(-33.096774193548384, -0.79848164520468146),
'W-EA':(24.387096774193594, 1.8428516515197337),
'C-EA':(83, 1.7000768787238201),
'E-EA':(131.80645161290323, -1.5837428955822119)}

uqsegtxt2 = {'Ross':(-173, -3.2),
'Byrd':(-149, 3.7),
'Amundsen':(-123, -2.6),
'Bellingshausen':(-95, -2.7),
'S-AP':(-70.258064516129025, 3.85),
'N-AP':(-59.225806451612897, 3.85),
'Weddell':(-33.096774193548384, -2.1),
'W-EA':(25.5, 3.3),
'C-EA':(83, 3.1),
'E-EA':(152, -1.5837428955822119)}

uqsegtxt3 = {'Ross':(-166, 0.75),
```

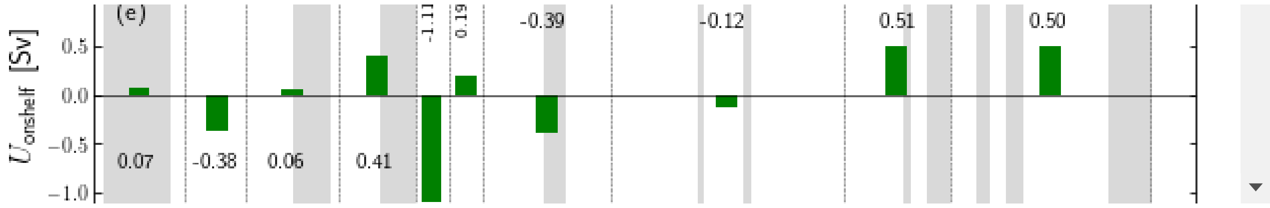
```
        'Byrd':(-140, 0.75),
        'Amundsen':(-117, 0.75),
        'Bellingshausen':(-88, 0.75),
        'S-AP':(-70.25806451612902 5, 0.75),
        'N-AP':(-59.225806451612897, 0.75),
        'Weddell':(-33.096774193548384, 0.75),
        'W-EA':(25.5, 0.75),
        'C-EA':(83, 0.75),
        'E-EA':(132, 0.75)}

        # Get along-shelf heat transport divergences.
        if GET_HT_DIVERGENCE:
            #========================================================
            fnamez = head+'hflxmelt_alongshelf_xwbry_tseries%dm.npz'%isob
            dashf = np.load(fnamez, encoding='bytes')
            segs = [b'N-AP', b'E-EA', b'Weddell', b'W-EA', b'S-AP', b'Byrd', b'Am
            t = dashf['t']
            UQx_timemean_alongshelf_convergence = dict()
            for seg in segs_lims.keys():
                uqxconv_ashf = -dashf['UQxdiv_ashf'].flatten()[0][seg].mean()  #
                UQx_timemean_alongshelf_convergence.update({seg:uqxconv_ashf}) #
            #========================================================

            nseg=1
            barw = 3
            for seg in segs_lims.keys():
                ax[3].bar(lonpltsl[seg], uqxmseg_avgs[seg], barw, linewidth=0, co
                ax[3].bar(lonpltsc[seg], uqxeseg_avgs[seg], barw, linewidth=0, co
                ax[3].bar(lonpltsr[seg], uqxseg_avgs[seg], barw, linewidth=0, col
                #
                # Plot total heat transport in segment as text.
                if seg in ['Ross', 'Byrd', 'Amundsen', 'Bellingshausen']:
                    ha = 'left'
                elif seg=='Weddell':
                    ha = 'right'
                else:
                    ha = 'center'
                xtt, ytt = uqsegtxt[seg]
                xtt2, ytt2 = uqsegtxt2[seg]
                if seg in ['S-AP', 'N-AP']:
                    fontsize = 8.5
                    rotation = 90
                elif seg=='Weddell':
                    fontsize = 10.5
                    rotation = 0
                else:
                    fontsize = 12
                    rotation = 0
                ax[3].text(xtt, ytt, round(uqxseg_total[seg], 1), color='k', font
                txt2 = '(' + str(round(UQx_timemean_alongshelf_convergence[seg],
                ax[3].text(xtt2, ytt2, txt2, color='k', fontsize=(fontsize-2), fo
                if nseg==1:
```

```
                ax[3].axhline(y=0, color='k', linewidth=0.6, linestyle='-')
                legh = ax[3].legend(loc=(0.5, 0.02), frameon=False, fancybox=
                for legobj, legtxt in zip(legh.legendHandles, legh.get_texts(
                    legtxt.set_color(legobj.get_facecolor())
            nseg+=1

        np.savez(head+'hflxmelt_onshelf_tseries.npz', uqxseg_onshore=uqxseg_t

        f_ashf_div = head+'hflxmelt_alongshelf_xwbry_tseries%dm.npz'%isob
        dd_ashf = np.load(f_ashf_div, encoding='bytes')
        UQxdiv_ashf = dd_ashf['UQxdiv_ashf'].flatten()[0]
        Ux_ashf = dd_ashf['Ux'].flatten()[0]
        Uxdiv_ashf = dd_ashf['Uxdiv_ashf'].flatten()[0]

        segnames = ['C-EA', 'W-EA', 'Ross', 'Byrd', 'S-AP', 'Weddell', 'Amund
        bsegnames = [b'C-EA', b'W-EA', b'Ross', b'Byrd', b'S-AP', b'Weddell',
        uqxseg_total_conv = dict()
        uxseg_total_conv = dict()
        uxseg_ashf_conv = dict()
        for segn in segnames:
            uqtotconv = uqxseg_total[segn] - UQxdiv_ashf[segn]
            utotconv = uxseg_total[segn] - Uxdiv_ashf[segn]
            uqxseg_total_conv.update({segn:uqtotconv})
            uxseg_total_conv.update({segn:utotconv})
            uxseg_ashf_conv.update({segn:-Uxdiv_ashf[segn]}) # Just switch si

        np.savez(head+'hflxmelt_convergence_tseries.npz', uqxseg_onshore=uqxs

    nseg=1
    barw = 7
    for seg in segs_lims.keys(): # Volume transport integrated over each segm
        xtt3, ytt3 = uqsegtxt3[seg]
        uxseg_onsh = uxseg_total[seg].mean()
        txt3 = '%1.2f'%uxseg_onsh # Onshore transport only.
        if seg in ['S-AP', 'N-AP']:
            ax[4].text(xtt3, ytt3, txt3, fontsize=9, fontweight='black', ha='
        elif seg in ['Ross', 'Byrd', 'Amundsen', 'Bellingshausen']:
            ax[4].text(xtt3, -0.7, txt3, fontsize=11, fontweight='black', ha=
        else:
            ax[4].text(xtt3, ytt3, txt3, fontsize=11, fontweight='black', ha=

        if nseg==1:
            ax[4].bar(lonpltsc[seg], uxseg_onsh, barw, linewidth=0, color='g'
        else:
            ax[4].bar(lonpltsc[seg], uxseg_onsh, barw, linewidth=0, color='g'
        nseg+=1

    ax[4].axhline(y=0, color='k', linewidth=0.6, linestyle='-')
    ax[3].set_ylabel(r'HT density [MW/m]', fontsize=14, fontweight='black', l
    ax[4].set_ylabel(r'$U_\text{onshelf}$ [Sv]', fontsize=15, fontweight='bla

    for txt, xtxt in seg_txts.items():
```

```python
        if txt=='Amundsen' or 'EA' in txt:
            yt = -61
        elif txt=='S-AP' or txt=='N-AP':
            yt = -57.8
        elif txt=='Bellingshausen':
            yt = -58.5
        else:
            yt = -65.0

        if 'EA' in txt or txt=='Weddell' or txt=='Ross' or txt=='Byrd':
            rotation = 0
        else:
            rotation = 90

        if txt=='S-AP' or txt=='N-AP':
            fontsize = 7.5
        elif txt=='Amundsen' or txt=='Bellingshausen':
            fontsize=11
        else:
            fontsize = 11.5

        ax[5].text(xtxt, yt, txt, fontsize=fontsize, fontweight='demibold', r
        if txt=='Ross':
            ax[5].text(173, yt, txt, fontsize=fontsize, fontweight='demibold'

    tt3 = 'Time- and segment-averaged $\Phi$'
    xletter, yletter = 0.022, 0.825
    xtt, ytt = 0.065, 0.825
    ax[0].text(xletter, yletter, r'(a)', fontsize=13, fontweight='bold', tran
    ax[1].text(xletter, yletter, r'(b)', fontsize=13, fontweight='bold', tran
    ax[2].text(xletter, yletter, r'(c)', fontsize=13, fontweight='bold', tran
    ax[3].text(xletter, yletter, r'(d)', fontsize=13, fontweight='bold', tran
    ax[4].text(xletter, yletter, r'(e)', fontsize=13, fontweight='bold', tran
    ax[5].text(xletter, yletter, r'(f)', fontsize=13, fontweight='bold', tran

    fig.subplots_adjust(hspace=0.0)#0.04)#0.025)
    legh = ax[2].legend(loc=(0.10, 0.72), frameon=False, fancybox=False, fram

    for legobj, legtxt in zip(legh.legendHandles, legh.get_texts()):
        legtxt.set_color(legobj.get_color())

    ax[2].grid(False)
    ax[1].grid(False)
    ax[0].grid(False)
    ax[3].grid(False)
    ax[4].grid(False)
    ax[5].grid(False)
    ax[5].set_xlim((-180, 180))

    # Add shadings marking segments of O(100-1000 km) onshore HT.
    darkgray = [0.85]*3
    rects = [(-115.00, -102.34), (-86.00, -75.00), (-32.90, -25.645), (17.419
```

```
              (32.18, 35.08), (84.68, 87.097), (108.63, 112.99), (118.31, 124.
              (-176.52, -154.45), (92.32, 100.45)]
    for rect in rects:
        for a in [0, 1, 2, 3, 4, 5]:
            if a==5:
                zorder = -10
            else:
                zorder = -9999
            ax[a].axvspan(*rect, facecolor=darkgray, alpha=1, zorder=zorder)

    kwtxt0 = dict(fontsize=18, fontweight='black', transform=ax[0].transAxes)
    ax[0].text(0.05, 0.05, r'$\Phi^\text{mean}$', color='b', **kwtxt0)
    ax[0].text(0.15, 0.05, r'$\Phi^\text{eddy}$', color='r', **kwtxt0)
    kwtxt1 = dict(fontsize=16, fontweight='black', transform=ax[1].transAxes)
    ax[1].text(0.025, 0.47, r'$\Phi^\text{mean}$', color='b', **kwtxt1)
    ax[1].text(0.110, 0.47, r'$\Phi^\text{eddy}$', color='r', **kwtxt1)
    ax[1].text(0.200, 0.47, r'$\Phi$', color='k', **kwtxt1)

    _ = [axn.set_facecolor('w') for axn in ax.tolist()]
    plt.draw()
    plt.show()

    figname = 'fig_circphi_tmean.png'
    fig.savefig(figname, bbox_inches='tight', pad_inches=0.0, dpi=125)
    plt.show()
```





(e)
U_{onshelf} [Sv]
0.5
0.0
−0.5
−1.0
0.07
-0.38
0.06
0.41
-1.11
0.19
-0.39
-0.12
0.51
0.50