Supplementary Material for: "Short Commentary on Marine Productivity at Arctic Shelf Breaks: Upwelling, Advection and Vertical Mixing"

Achim Randelhoff & Arild Sundfjord

March 13 2018

Description of the algorithm

The code below reproduces the shelf break depths plotted in Fig. 2 of the main paper. Briefly, the steps are

1. Prior to running the python script, a shapefile "shelfbreak.shp" had been created in a GIS application. It contains two contours outlining the area where the shelfbreak is located, see Fig. 1.

2. Going along the upper of the two contours, for each of its points the closest point on the other contour is located. Each pair of points defines a cross-shelf slope transect. Transects are cleaned by setting a maximum depth of 700 m. Furthermore, the Saint Anna Trough in the Kara Sea and the Chukchi Borderland off the Chukchi Sea are omitted from this analysis due to their difficult topography that does not easily allow the definition of a clear shelf break.

3. Each of these transects is then used in a nonlinear least-squares fit. The fitting function is defined to be piecewise linear, consisting of two pieces, joined by a breakpoint. This breakpoint will define the shelf break. See Figs. 4–**??**

4. Based on histograms of the resulting slopes both on- and off-shelf, criteria are set to further filter out ambiguous shelf breaks:

    - The shelf slope has to be at least 10 m/km steep
    - The absolute value of slope of the shelf itself must not exceed 5 m/km
    - The shelf slope has to be at least four times as steep as the on-shelf slope

All transects and fits were visually checked for agreement with the intuitive definition of the term "shelf break" and for accuracy of the fitting routine.

The last step (number 4 above) does in particular remove a number of "step-like" profiles, with upper and lower "shelf breaks". The lower of these would represent the actual shelf break, i.e. the northward extent of the shelf, while the upper one could in some instances potentially be dynamically relevant for wind-driven surface circulation, and hence for upwelling.

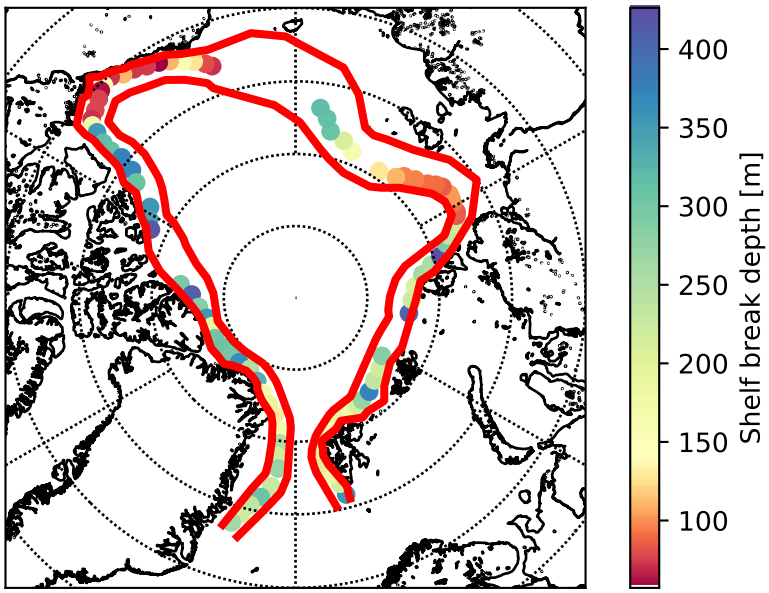

Figure 1: Another version of Fig. 2 of the main paper, this time showing the outlines used to build the transects to search the shelf break (red, thick lines).

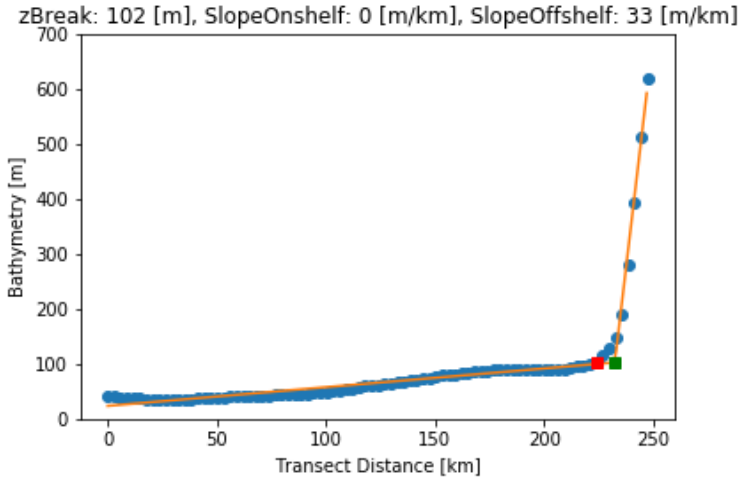

Figure 2: Well-behaved transect with a clear shelf break (122.88 °E, 77.15 °N). Shown are the the transect (blue dots), piecewise linear best-fit regression lines (orange lines), the initial guess for the shelf break (red) and the final detected shelfbreak (green).

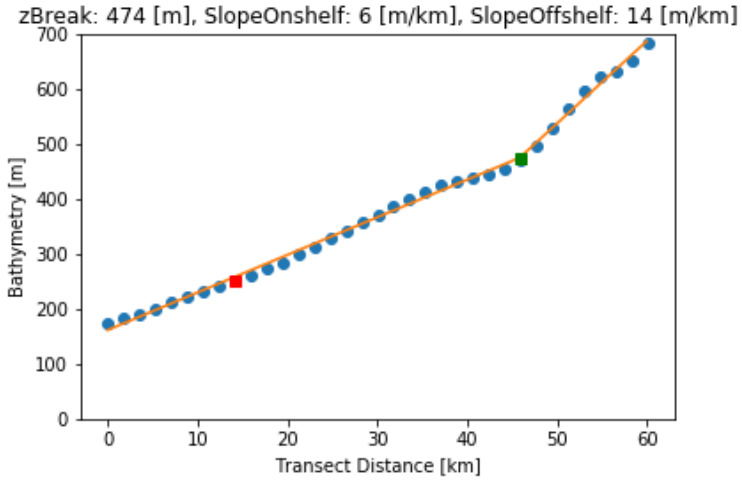

Figure 3: Same as above, but a very gentle slope that does not fulfill the steepness criteria, see point 4 above (128.81 °W, 73.21 °N)

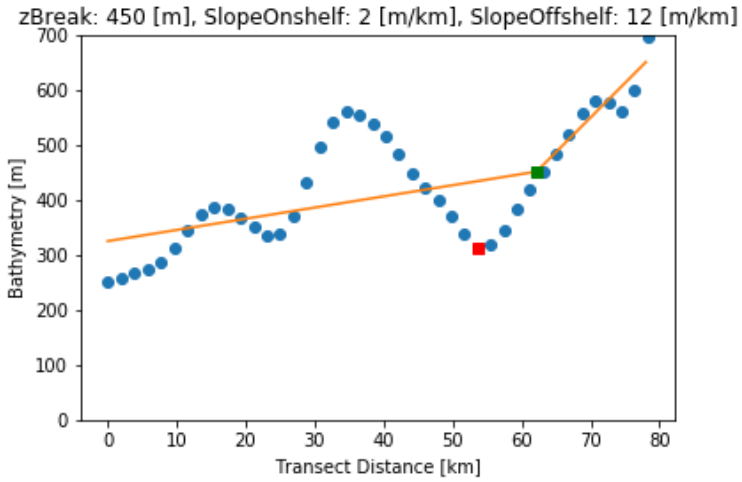

Figure 4: Same as above, but a "pathological" profile that does not permit the definition of a shelf break in the present sense, and indeed it is also filtered out based on steepness criteria (118.81 °W, 78.54 °N)

Python code

```
import numpy as np

#%%
# Define bin average for later smoothing of shelf
    break plot
def BinAverage (Data,WindowLength):
    BinAveragedData = []
    for k in range(0,np.int(len(Data)/WindowLength)):
        thedata = Data[k*WindowLength:(k+1)*
            WindowLength]
        if np.all(np.isnan(thedata)):
            BinAveragedData.append(np.nan)
        else:
            BinAveragedData.append(np.nanmean(thedata)
                )
    return np.array(BinAveragedData)

#%%
# Define a class for the re-normalization of the color
    scale (later on)
# modified from https://matplotlib.org/users/
    colormapnorms.html
import matplotlib.colors as colors
class MidpointNormalize(colors.Normalize):
    def __init__(self, vmin=None, vmax=None, midpoint=
        None, clip=False):
        self.midpoint = midpoint
        colors.Normalize.__init__(self, vmin, vmax,
            clip)

    def __call__(self, value, clip=None):
        # I'm ignoring masked values and all kinds of
            edge cases to make a
        # simple example...
        x, y = [self.vmin, self.midpoint, self.vmax],
            [0, 0.5, 1]
        return np.ma.masked_array(np.interp(value, x,
            y))


#%%
```

```
# Import rough outline of shelfbreak, previously drawn
    using the QGIS software
# One outline runs 'above' the assumed shelfbreak, one
    below. This will be used
# as boundaries of cross-shelf slope transects

import shapefile as shp
TransectOutlineShapefile = shp.Reader('qgis/shelfbreak
    .shp').shapes()

#%%
# load Basemap and define a stereographic projection

import mpl_toolkits.basemap as bm
m = bm.Basemap(projection='npstere', boundinglat=70,
    lon_0=0, resolution='h')

#%%
# Increase along-track resolution of the outlines
# (to increase resolution of across-shelf transects
    for shelf break search)

# original outlines
Outline_Coord_FromShapefile = [[],[]] # coordinates
Outline_Distance_FromShapefile = [[],[]] # along-track
    distance

# new, interpolated outline
Outline_Distance_Interpolated=np.arange(0,15e6,10e3) #
    same for all outlines
Outline_Coord_Interpolated = [[[],[]],[[],[]]]

for i in range(0,2):
    Outline_Coord_FromShapefile[i] = m(np.array(
        TransectOutlineShapefile[i].points)[:,0],np.
        array(TransectOutlineShapefile[i].points)[:,1])
    Outline_Distance_FromShapefile[i] = np.r_[0,np.
        cumsum(
            np.sqrt(np.diff(
                Outline_Coord_FromShapefile[i][0])**2 +
                 np.diff(Outline_Coord_FromShapefile[i
                ][1])**2)
            )]
    for j in range(0,2):
```

```
            Outline_Coord_Interpolated[i][j] = np.interp(
                Outline_Distance_Interpolated,
                                    Outline_Distance_FromShapefile
                                        [i],
                                        Outline_Coord_FromShapefile
                                        [i][j])
            Outline_Coord_Interpolated[i][j][
                Outline_Distance_FromShapefile[i].max()
            < Outline_Distance_Interpolated] = np.nan

        Outline_Coord_Interpolated[i] = np.c_[
            Outline_Coord_Interpolated[i][:]]

#%%
# now go along one of the contours (
    indexOutline_TransectStart) and find closest
    neighbour on other contour (
    indexOutline_TransectEnd) - that will define our
    transect

# start looking from this contour...
indexOutline_TransectStart=0
# ... and find nearest neighbor on this contour
indexOutline_TransectEnd=1

def lookupNearest (k):
    norm=np.linalg.norm(Outline_Coord_Interpolated[
        indexOutline_TransectEnd].transpose()
         -Outline_Coord_Interpolated[
             indexOutline_TransectStart].transpose()[k],
             axis=1)
    if np.all(np.isnan(norm)):
        return np.array([np.nan,np.nan])
    else:
        return Outline_Coord_Interpolated[
            indexOutline_TransectEnd][:,np.nanargmin(
            norm)]



# initialize a vector that holds start and end points
    of all transects
Transects_StartEndPoints = [np.zeros_like(
    Outline_Coord_Interpolated[
```

```
    indexOutline_TransectStart]),
                np.zeros_like(
                    Outline_Coord_Interpolated[
                    indexOutline_TransectStart])]
# populate vector
for k in range(0,len(Outline_Distance_Interpolated)):
    Transects_StartEndPoints[
        indexOutline_TransectStart][:,k] =
        Outline_Coord_Interpolated[
        indexOutline_TransectStart][:,k]
    Transects_StartEndPoints[indexOutline_TransectEnd
        ][:,k] = lookupNearest(k)
# convert the start and end coordinates into lat/lon
    for interpolation and filter out nans
for ii in range(0,2):
    Transects_StartEndPoints[ii] = np.c_[m(*
        Transects_StartEndPoints[ii],inverse=True)]
    Transects_StartEndPoints[ii][
        Transects_StartEndPoints[ii]==1e30]=np.nan

Transects_StartEndPoints[indexOutline_TransectEnd] =
    Transects_StartEndPoints[indexOutline_TransectEnd
    ][~np.isnan(Transects_StartEndPoints[i][:,0]),:]
Transects_StartEndPoints[indexOutline_TransectStart] =
     Transects_StartEndPoints[
    indexOutline_TransectStart][~np.isnan(
    Transects_StartEndPoints[indexOutline_TransectStart
    ][:,0]),:]

#%%
# Load IBCAO bathymetry
import scipy.io as sio
top = sio.loadmat('/Users/doppler/data/IBCAO/
    IBCAO_1min_bathy.mat');
LonMap,LatMap = top['x'][0],top['y'][0]
ZMap=-top['IBCAO_1min'][:,:]

# define interpolating function to extract bathymetry
    at arbitrary coordinates
import scipy.interpolate as interp
interpolating_function = interp.
    RegularGridInterpolator((LatMap,LonMap),ZMap)

#%%
```

```
# for kth point of along-shelf contour, define a
    transect (Distance+Bathymetry)
# towards the nearest neighbour on the neighbouring
    contour

def extract_transect(k):
    thislat = np.interp(np.linspace(0,1,100,endpoint=
        True),
                        np.array([0,1]),[
                            Transects_StartEndPoints[
                            indexOutline_TransectStart][
                            k,1],
                            Transects_StartEndPoints[
                            indexOutline_TransectEnd][k
                            ,1]])
    thislon = np.interp(np.linspace(0,1,100,endpoint=
        True),
                        np.array([0,1]),[
                            Transects_StartEndPoints[
                            indexOutline_TransectStart][
                            k,0],
                            Transects_StartEndPoints[
                            indexOutline_TransectEnd][k
                            ,0]])
    thisx,thisy=m(thislon,thislat)
    thisd = np.linalg.norm(np.c_[thisx-thisx[0],thisy-
        thisy[0]],axis=1) /1e3
    thisz = interpolating_function(np.c_[thislat,
        thislon])
    # return vectors of:
    # along-transect distance d, bathymetry z,
        longitude lon, and latitude lat
    return thisd, thisz, thislon, thislat

# define a piecewise linear function to be used when
    fitting cross-shelf bathymetry transect
def piecewise_linear(x, x0, y0, k1, k2):
    return np.piecewise(x, [x < x0], [lambda x:k1*x +
        y0-k1*x0, lambda x:k2*x + y0-k2*x0])

from scipy import optimize

#%% Given a certain transect bathymetry, calculate the
    shelf break
```

```
# as the break point of a piecewise linear function
    consisting of two pieces.
# We do this using the optimize.curve_fit function
#
# input: along-track distance d, bathymetry z,
# [optional, default 100 m] Initial guess for shelf
    break depth
# [optional, default 1000 m] Deepest depth to be
    considered for fitting of cross-shelf transect
import matplotlib.pyplot as plt
#import os
def find_breakpoint (d,z,k, ReferenceDepthDeep,
    ReferenceDepthBreak=400):
    # mask out everything that follows after profile
        drops below maximum depth specified above
    if np.nanmax(z)>ReferenceDepthDeep:
        mask=np.s_[np.argwhere(z>ReferenceDepthDeep)
            [0][0]:]
        d=np.delete(d,mask)
        z=np.delete(z,mask)
    # find initial guess for breakpoint:
    try:
        ReferenceDepthBreak = np.max([np.nanmin(z)+5,
            ReferenceDepthBreak])
        # shallowest point in 20 km vicinity of
            ReferenceDepthBreak (default 400 m) point
        id_ReferenceDepthBreak = np.max(np.where(z<
            ReferenceDepthBreak))
        id_init = np.where(z==np.nanmin(z[
                (d[id_ReferenceDepthBreak]-20<=d) & (d
                    <=d[id_ReferenceDepthBreak]+20)
                ]))[0][0]

        zInit=z[id_init]
        dInit=d[id_init]
        # initialize fitting function with this guess
        def piecewise_linear_init(x,k1,k2):
            return piecewise_linear(x,dInit,zInit,k1,
                k2)

        if np.sum(~np.isnan(z))>=4:
            # optimize for complete initial guess to
                be plugged into the actual optimization
            p0_Rest , e = optimize.curve_fit(
```

```
                piecewise_linear_init, d, z)
            p0 = np.array([dInit, zInit, p0_Rest[0],
                p0_Rest[1]])
            # do actual optimization
            p, e = optimize.curve_fit(
                piecewise_linear, d, z, p0=p0)

            # plotting each Transect for diagnostic
                purposes
            if False:
                dd = np.arange(0, d.max(),1)
                plt.plot(d, z, "o")
                plt.plot(dd, piecewise_linear(dd, *p))
                plt.ylim(0,ReferenceDepthDeep)
                plt.title('zBreak: %i [m], '
                    'SlopeOnshelf: %i [m/km], '
                    'SlopeOffshelf: %i [m/km]'
                        % (p[1], p[2], p[3]))
                plt.plot(dInit, zInit, 'rs')
                plt.plot(p[0],p[1],'gs')
                plt.xlabel('Transect Distance [km]')
                plt.ylabel('Bathymetry [m]')
                plt.savefig(
                        'transect_plots/transect_%04i.
                            png'
                        % k)
                plt.close()
        else:
            p=np.array([np.nan,np.nan,np.nan,np.nan])
    except:
            p=np.array([np.nan,np.nan,np.nan,np.nan])
    return p

#%%
# calculate regression for kth index of the Transects
def calculate_shelfbreak (k,ReferenceDepthDeep):
    # extract the transect distance d, bathymetry z,
        and lat/lon coordinates
    d,z,lon,lat=extract_transect (k)
    # if transect is non-empty, find the breakpoint of
        a piecewise linear profile
    if len(z)==0:
        print('FAIL')
        lonBreak=np.nan
```

```
        latBreak=np.nan
        zBreak=np.nan
        d=np.nan
        z=np.nan
    else:
        p = find_breakpoint(d,z,k,ReferenceDepthDeep)
        # translate to shelf break...
        lonBreak = np.interp(p[0],d,lon)
        latBreak = np.interp(p[0],d,lat)
        zBreak = p[1]
        SlopeOnshelf = p[2]
        SlopeOffshelf = p[3]

    return lonBreak,latBreak,zBreak,d,z,SlopeOnshelf,
        SlopeOffshelf

#%%
# Now comes the actual loop along the entire shelf
    break outlines

# initialize variables for loop
LON=[]
LAT=[]
ZBreak=[]
D=[]
Z=[]
S1=[]
S2=[]

# Transects / Depth profiles to loop over
ProfileNumber=np.arange(0,len(Transects_StartEndPoints
    [0]),1)
# Skip: Saint Anna Trough and Chukchi Borderlands
ProfileNumberSkip = np.r_[np.arange(150,181),np.arange
    (460,551)]
# Set reference depth that will limit the maximum
    allowed depth to be included in shelf break
    detection routine
ReferenceDepthDeep=700

for k in ProfileNumber:#,len(d)-1,10000):
    if np.in1d(k,ProfileNumberSkip):
        lon,lat,zBreak,d,z,s1,s2=np.zeros([7,1])*np.
            nan
```

```
        else:
            lon,lat,zBreak,d,z,s1,s2=calculate_shelfbreak(
                k,ReferenceDepthDeep)
        LON.append(lon)
        LAT.append(lat)
        ZBreak.append(zBreak)
        D.append(d)
        Z.append(z)
        S1.append(s1)
        S2.append(s2)

ShelfbreakLon=np.array(LON)
ShelfbreakLat=np.array(LAT)
Shelfbreakz=np.array(ZBreak)
TransectDistance = D
Transectz = Z
ShelfbreakX,ShelfbreakY=m(ShelfbreakLon,ShelfbreakLat)
ShelfbreakX[np.isnan(ShelfbreakLon)] = np.nan
ShelfbreakY[np.isnan(ShelfbreakLon)] = np.nan
SlopeOnshelf = np.array(S1)
SlopeOffshelf = np.array(S2)


# %%
# Further filter detected shelf breaks based on:
# - the slope of the shelf itself should have max 5 m/
    km
# - the slope of the shelf slope should be steeper
    than 10 m/km
# - the shelf slope has to be at least four times as
    steep as the on-shelf slope
# These thresholds were derived based on overall
    detected slopes
Filter = (np.abs(SlopeOnshelf)>5) | (SlopeOffshelf<10)
    | (np.abs(SlopeOffshelf)<4*SlopeOnshelf)
ShelfbreakzFiltered = np.where(Filter,np.nan,
    Shelfbreakz)
NumberDatapointsFiltered = np.sum(Filter)
print('Filtered: %i datapoints' %
    NumberDatapointsFiltered)

Window = 10
```

```
#%%
# Make plot

# make colormap
cmap = plt.cm.get_cmap('bone')
cmap.set_under('white')
cmap.set_over('white')

fig,ax=plt.subplots()

# LABELS
m.drawparallels(np.arange(60,100,5))
m.drawmeridians(np.arange(0,360,60),labels=[True,False
    ,True,True])
TheLabels = ['65','75','80','85']
TheLon = np.array([1,1,1,1])*(-135)
TheLat = [66,76,81,86]
for k in range(len(TheLabels)):
    plt.text(*m(TheLon[k],TheLat[k]),TheLabels[k]+r'$
        ^\circ$N')

# plot same bathymetry as in other plot, for reference
hcs1=ax.contourf(*m(*np.meshgrid(LonMap,LatMap)),ZMap,
                 levels=[0,75,250,500],cmap=cmap,
                 extend='both')
m.drawcoastlines(color='k',linewidth=1)

hcs2 = m.scatter(
        BinAverage(ShelfbreakX,Window),
        BinAverage(ShelfbreakY,Window),
        c=BinAverage(ShelfbreakzFiltered,Window),
        marker='o',
        norm=MidpointNormalize(midpoint=140.),#vmin=0,
        cmap='Spectral'#'PiYG'
        )
hcb1 = plt.colorbar(hcs1,shrink=1)
hcb2 = plt.colorbar(hcs2)
hcb1.set_label('Bathymetry [m]')
hcb2.set_label('Shelf break depth [m]')

plt.savefig('../latex/map_bathy_shelfbreak.pdf',
    bbox_inches='tight')
```