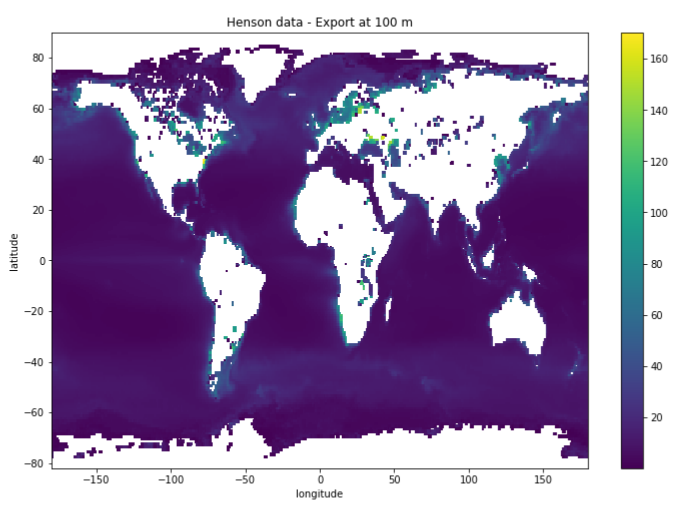
In [1]:
```
# import libraries
import scipy.io
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
from scipy.interpolate import interpn
import utm
import shapely
from pyproj import Proj
from shapely.geometry import shape
from shapely import geometry
import shapely.ops as ops
from shapely.geometry.polygon import Polygon
from functools import partial
import pyproj
```

Henson's data

In [2]:
```
henson = scipy.io.loadmat('expHensonmod.mat')
latH = np.concatenate(henson['lat'])
lonH = np.concatenate(henson['lon'])
export = henson['exportC'] # units in gC/m2/year
```

In [3]:
```
# plot carbon export at 100 m
fig, ax = plt.subplots(1, 1, figsize=(12,8))
im = ax.pcolormesh(lonH, latH, export, shading = 'auto')
fig.colorbar(im, ax=ax)
ax.set_title('Henson data - Export at 100 m')
ax.set_xlabel('longitude')
ax.set_ylabel('latitude')
#plt.savefig('./maps/henson_raw.jpg')
```

Out[3]: Text(0, 0.5, 'latitude')



Tim's data

In [4]:
```
# extract and clean a bit.
tim = scipy.io.loadmat('F100.mat')
pc_co2 = tim['F100'] # % of sequestered carbon
latT = np.concatenate(tim['lat'])
lonT = np.concatenate(tim['lon'])
depthT = np.concatenate(tim['depth'])

# Tim's longitude are eastward so from 0 to 360. I want -180 to 180. Hence, a bit of 
pc_co2 = np.concatenate((pc_co2[:,90:,:], pc_co2[:,:90,:]), axis = 1)
lonT = np.linspace(-179, 179, 180)

# replace 1 by 0 in Tim's first layer because it should not be 100% of sequestration 
pc_co2[:,:,0] = np.where(pc_co2[:,:,0] == 1 , 0, pc_co2[:,:,0])
```

In [5]:
```
# plot f100 for each depth
def plot_tim(lat, lon, index_depth, depth, data):
    fig, ax = plt.subplots(1, 1, figsize=(12,8))
    im = ax.pcolormesh(lon, lat, data[:,:,index_depth], vmin = 0, vmax = 100, shading
    fig.colorbar(im, ax=ax)
    d = int(np.floor(depth[index_depth]))
    ax.set_title('% of captured CO2 at '+str(d)+ ' m')
    ax.set_xlabel('longitude')
    ax.set_ylabel('latitude')
    d = int(np.floor(depth[index_depth]))
    plt.savefig('./maps/Tim/f100_'+str(d)+'_m.jpg')
    plt.close()
```

In [6]:
```
for i, depth in enumerate(depthT):
    plot_tim(latT, lonT, i, depthT, pc_co2*100)
```

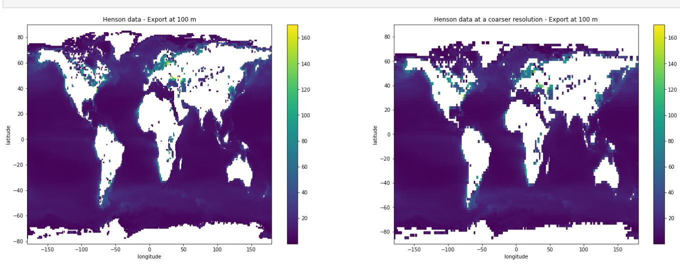
Interpolate Henson's data (though, it's a finer grid) on Tim's data

In [7]:
```
RlatH = latH[::-1] # reverse latH for ascending order needed by scipy

points = (RlatH, lonH)
values = export

RlatT = latT[::-1]
RlonT = lonT[::-1]
```

In [8]:
```
coarser_export = []
for i, lat in enumerate(RlatT):
    for j, lon in enumerate(lonT):
        point = np.array([lat,lon])
        try:
            tmp = float(interpn(points, values, point, method = 'nearest'))
            coarser_export.append(tmp)
        except:
            tmp = np.nan
            coarser_export.append(tmp)

coarser_export = np.array(coarser_export).reshape((91, 180)) # shape of pc_co2 (Tim's
print(np.nanmax(coarser_export))
print(np.nanmean(coarser_export))
```

```
170.24612426757812
11.490637334499272
```

In [9]:
```
# plot both resolutions (just to check if it's OK)
fig, (ax0, ax1) = plt.subplots(1, 2, figsize=(24,8))
im0 = ax0.pcolormesh(lonH, latH, export, shading = 'auto')
fig.colorbar(im0, ax=ax0)
ax0.set_title('Henson data - Export at 100 m')
ax0.set_xlabel('longitude')
ax0.set_ylabel('latitude')

im1 = ax1.pcolormesh(lonT, latT, coarser_export, shading = 'auto')
fig.colorbar(im1, ax=ax1)
ax1.set_title('Henson data at a coarser resolution - Export at 100 m')
ax1.set_xlabel('longitude')
ax1.set_ylabel('latitude')

plt.savefig('./maps/henson_resolution_comparison.jpg')
```

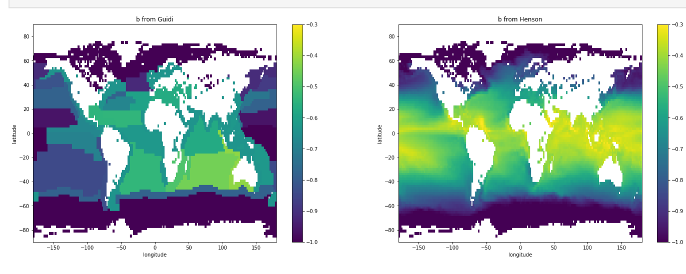


Martin's coefficient (i.e. b) + interpolation on Tim's grid

In [10]:
```
# Martin's coefficient from Lionel
martin = scipy.io.loadmat('B_martin.mat')
b_guidi = martin['B_Guidi']
b_henson = martin['B_Henson']
```

In [11]:
```
# interpolation on coarser grid
b_guidi_coarse = []
b_henson_coarse = []
for i, lat in enumerate(RlatT):
    for j, lon in enumerate(lonT):
        point = np.array([lat,lon])
        try:
            tmp_guidi = float(interpn(points, b_guidi, point, method = 'nearest'))
            b_guidi_coarse.append(tmp_guidi)
        except:
            tmp_guidi = np.nan
            b_guidi_coarse.append(tmp_guidi)
        try:
            tmp_henson = float(interpn(points, b_henson, point, method = 'nearest'))
            b_henson_coarse.append(tmp_henson)
        except:
            tmp_henson = np.nan
            b_henson_coarse.append(tmp_henson)

b_guidi_coarse = np.array(b_guidi_coarse).reshape((91, 180)) # shape of pc_co2 (Tim's
b_henson_coarse = np.array(b_henson_coarse).reshape((91, 180)) # shape of pc_co2 (Tim
```

In [12]:
```
# plot it
fig, (ax0, ax1) = plt.subplots(1, 2, figsize=(24,8))
im0 = ax0.pcolormesh(lonT, latT, b_guidi_coarse, shading = 'auto', vmin = -1, vmax= -(
fig.colorbar(im0, ax=ax0)
ax0.set_title('b from Guidi')
ax0.set_xlabel('longitude')
ax0.set_ylabel('latitude')


im1 = ax1.pcolormesh(lonT, latT, b_henson_coarse, shading = 'auto', vmin = -1, vmax =
fig.colorbar(im1, ax=ax1)
ax1.set_title('b from Henson')
ax1.set_xlabel('longitude')
ax1.set_ylabel('latitude')

plt.savefig('./maps/b_coarse_Guidi_Henson.jpg')
```



Computation of the remineralisation flux

In [13]:
```
depthT # depth levels from Tim's data
```

Out[13]:
```
array([4.93454087e+00, 1.48774788e+01, 2.51158413e+01, 3.59450532e+01,
       4.76605389e+01, 6.05577232e+01, 7.49320307e+01, 9.10788861e+01,
       1.09293714e+02, 1.29871939e+02, 1.53108986e+02, 1.79300279e+02,
       2.08741243e+02, 2.41727303e+02, 2.78553883e+02, 3.19516408e+02,
       3.64910303e+02, 4.15030992e+02, 4.70173900e+02, 5.30634451e+02,
       5.96708071e+02, 6.68690183e+02, 7.46876213e+02, 8.31561585e+02,
       9.23041723e+02, 1.02161205e+03, 1.12756800e+03, 1.24120499e+03,
       1.36281844e+03, 1.49270378e+03, 1.63115644e+03, 1.77847183e+03,
       1.93494539e+03, 2.10087254e+03, 2.27654870e+03, 2.46226930e+03,
       2.65832976e+03, 2.86502551e+03, 3.08265197e+03, 3.31150457e+03,
       3.55187872e+03, 3.80406987e+03, 4.06837342e+03, 4.34508481e+03,
       4.63449945e+03, 4.93691279e+03, 5.25262023e+03, 5.58191720e+03])
```

In [14]:
```
def compute_Fseq_prime(z1, z2, b, f_100, Fexp, Zexp = 100):
    """
    INPUTS : z1, z2 : depths above and below depth of interest
             b : Martin's coefficient
             f_100 : % of captured carbon
             Fexp : flux of exported carbon
             Zexp : depth of export
    OUTPUT : Fseq_prime : the flux of sequestered carbon
    """
    if ((np.isnan(b)) | (np.isnan(f_100)) | (np.isnan(Fexp))):
        Fseq_prime = np.nan
        delta_flux = np.nan
    else:
        delta_flux = Fexp * ((z1/Zexp)**b - ((z2/Zexp)**b))
        Fseq_prime = delta_flux * f_100 # units gC m-2 year-1
    return(Fseq_prime)
```

In [15]:
```
# Ok, let's loop it all, except for the surface and the bottom (missing either z1 or 

Fseq_prime = np.zeros(pc_co2.shape) # this will be for the 3D Fseq_prime

for i, depth in enumerate(depthT):
    for j, lat in enumerate(latT):
        for k, lon in enumerate(lonT):
            if((i == 0) | (i == len(depthT)-1)): # surface of bottom, we cannot compu
                r = int(np.array(np.where(latT == lat))) # r for row
                c = int(np.array(np.where(lonT == lon))) # c for column
                Fseq_prime[r,c,i] = np.nan
            else:
                r = int(np.array(np.where(latT == lat))) # r for row
                c = int(np.array(np.where(lonT == lon))) # c for column
                z1 = depthT[i-1]
                z2 = depthT[i+1]
                b = b_guidi_coarse[r,c]
                Fexp = coarser_export[r,c]
                f100 = pc_co2[r,c,i]
                Fseq_prime_z = compute_Fseq_prime(z1, z2, b, f100, Fexp)
                Fseq_prime[r,c,i] = Fseq_prime_z
```

Computation of pixel areas

In [16]:
```
# https://stackoverflow.com/questions/4681737/how-to-calculate-the-area-of-a-polygon-
# https://proj.org/operations/projections/aea.html
def pixel_area2(lon1, lon2, lat1, lat2):
    {"type": "Polygon",
     "coordinates": [[
       [lon2, lat2],
       [lon2, lat1],
       [lon1, lat1],
       [lon1, lat2]
     ]]}

    co = {"type": "Polygon", "coordinates": [
        [(lon2, lat2),
         (lon2, lat1),
         (lon1, lat1),
         (lon1, lat2)]]}
    lon, lat = zip(*co['coordinates'][0])

    pa = Proj("+proj=aea +lat_1="+str(lat1)+" +lat_2="+str(lat2)+" +lat_0="+str((lat2
    #pa = Proj("+proj=aea +lat_1="+str(lat1)+" +lat_2="+str(lat2))

    x, y = pa(lon, lat)
    cop = {"type": "Polygon", "coordinates": [zip(x, y)]}

    return(shape(cop).area*1e-6)  # in km2
```

In [17]:
```
# check formula with the Colorado (rectangle) state
lon1 = -109.05
lon2 = -102.05
lat1 = 37.
lat2 = 41.

print(pixel_area2(lon1, lon2, lat1, lat2)) # OK looks good
```

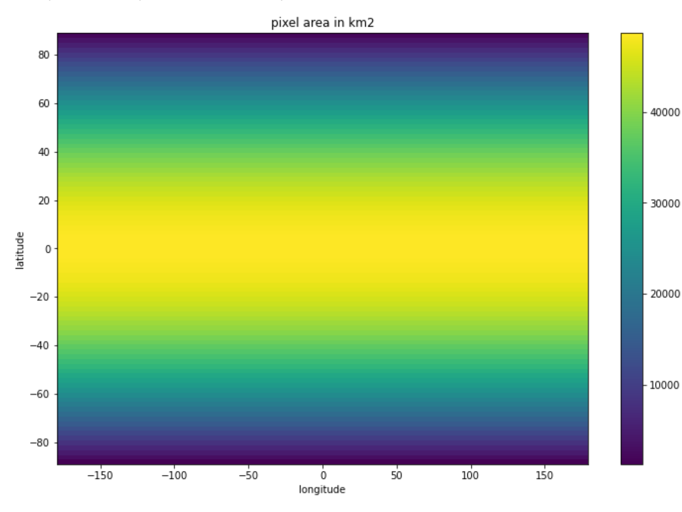
```
268952.04410743417
```

In [18]:
```
# how I computed the areas -> cheated a bit on the left and bottom edge, needs to be 

# pixel_results = np.zeros(b_guidi_coarse.shape)
# for i, lon in enumerate(lonT):
#     for j, lat in enumerate(latT):
#         if ((i == len(lonT)-1) | (j == len(latT)-1)):
#             lon1 = lonT[i]
#             lon2 = lonT[i] + 1
#             lat1 = latT[j]
#             lat2 = latT[j] + 0.9
#             pixel_results[j,i] = pixel_area2(lon1, lon2, lat1, lat2)
#         else:
#             lon1 = lonT[i]-0.5
#             lon2 = lonT[i+1]-0.5
#             lat1 = latT[j]-0.5
#             lat2 = latT[j+1]-0.5
#             pixel_results[j,i] = pixel_area2(lon1, lon2, lat1, lat2)
#             #print(pixel_area2(lon1, lon2, lat1, lat2))
```

In [19]:
```
# load pixel areas and plot it for check
pixel_areas = np.load('pixel_areas.npy')

fig, ax = plt.subplots(1, 1, figsize=(12,8))
im = ax.pcolormesh(lonT, latT, pixel_areas[:,:])
fig.colorbar(im, ax=ax)
ax.set_xlabel('longitude')
ax.set_ylabel('latitude')
ax.set_title('pixel area in km2')
```

```
<ipython-input-19-b87220e3490f>:5: MatplotlibDeprecationWarning: shading='flat' when X
and Y have the same dimensions as C is deprecated since 3.3.  Either specify the corne
rs of the quadrilaterals with X and Y, or pass shading='auto', 'nearest' or 'gouraud',
or set rcParams['pcolor.shading'].  This will become an error two minor releases late
r.
  im = ax.pcolormesh(lonT, latT, pixel_areas[:,:])
```

Out[19]: Text(0.5, 1.0, 'pixel area in km2')



In [20]:
```
# plot of sequestered carbon for each depth
def plot_fseq(lat, lon, index_depth, depth, data, pixel_area):
    fig, ax = plt.subplots(1, 1, figsize=(12,8))
    im = ax.pcolormesh(lon, lat, data[:,:,index_depth]*pixel_area*1e6*1e-15, vmin = 0,
    fig.colorbar(im, ax=ax)
    d = int(np.floor(depth[index_depth]))
    ax.set_title('Fseq_prime at '+str(d)+ ' m - Max Fseq = '+str(np.floor(np.nanmax(da
    ax.set_xlabel('longitude')
    ax.set_ylabel('latitude')
    d = int(np.floor(depth[index_depth]))
    plt.savefig('./maps/Tim_fseq/Fseq_'+str(d)+'_m.jpg')
    plt.close()

for i, depth in enumerate(depthT):
     plot_fseq(latT, lonT, i, depthT, Fseq_prime, pixel_areas)
```

```
<ipython-input-20-ac0d96939dd8>:7: RuntimeWarning: All-NaN slice encountered
  ax.set_title('Fseq_prime at '+str(d)+ ' m - Max Fseq = '+str(np.floor(np.nanmax(data
[:,:,index_depth]))))
```

Compute sequestration

In [21]:
```
print(depthT[7])
print(depthT[8])
```

```
91.07888607598377
109.29371391142902
```

In [22]:
```
above_100m_seq = np.nansum(Fseq_prime[:,:,:8], axis = 2)
below_100m_seq = np.nansum(Fseq_prime[:,:,8:48], axis = 2)

# show continents
above_100m_seq[above_100m_seq == 0] = 'nan'
below_100m_seq[below_100m_seq == 0] = 'nan'
```

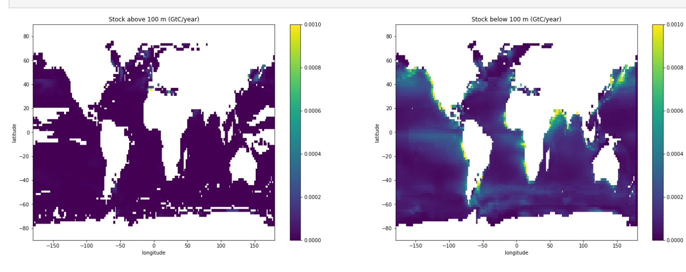
In [23]:
```
# Henson total sequestered carbon
np.nansum(np.nansum(coarser_export*pixel_areas[:,:]*1e6*1e-15, axis = 0)) # GtC year-
```

Out[23]: 4.093175434644271

In [24]:
```
print("Carbon sequestered above 100m (in Gtc/year): "+str(np.nansum(np.nansum(above_10
print("Carbon sequestered below 100m (in Gtc/year): "+str(np.nansum(np.nansum(below_10
```

```
Carbon sequestered above 100m (in Gtc/year): 0.05265522192948907
Carbon sequestered below 100m (in Gtc/year): 1.1352744355511333
```

In [25]:
```
fig, (ax0, ax1) = plt.subplots(1, 2, figsize=(24,8))
im0 = ax0.pcolormesh(lonT, latT, np.multiply(above_100m_seq[:,:],pixel_areas[:,:]*1e6
fig.colorbar(im0, ax=ax0)
ax0.set_title('Stock above 100 m (GtC/year)')
ax0.set_xlabel('longitude')
ax0.set_ylabel('latitude')

# henson
im1 = ax1.pcolormesh(lonT, latT, np.multiply(below_100m_seq[:,:],pixel_areas[:,:]*1e6
fig.colorbar(im1, ax=ax1)
ax1.set_title('Stock below 100 m (GtC/year)')
ax1.set_xlabel('longitude')
ax1.set_ylabel('latitude')
plt.savefig('./maps/sequestered_carbon.jpg')
```



In [26]:
```
# to do :

# deal with the edge issue for the pixel area
```