Exercices on Satellite Altimetry Data

Léna Tolu, Fabien Léger, Florence Birol, Fernando Niño

LEGOS/CTOH

Access to variables

We first import some libraries : datetime will allow us to better manipulate the time variables, matplotlib is used for the plots and cartopy for the maps. Numpy and pandas allow to manipulate data in general, while netCDF4 is designed for the treatment of netcdf files in particular. This notebook works with the following versions : python 3.9, shapely 2.0.1, Cartopy 0.21.1, matplotlib 3.7.2

In [75]:
```
import datetime
import numpy as np
import matplotlib.pyplot as plt
from netCDF4 import Dataset
import pandas as pd

import cartopy.crs as ccrs
import cartopy.feature as cpf
import warnings
warnings.filterwarnings("ignore")

%matplotlib notebook
```

We use Dataset to open and read the netcdf file. The data are classed by tracks. Here, we will work on track 1.

In [76]:
```
filename = 'dt_coastal_j3_phy_20hz_t001.nc'
f = Dataset(filename, 'r')
```

We can print the variables contained in the file :

In [77]:
```
for var in f.variables.keys():
    print(var)
```

```
latitude_theoretical
longitude_theoretical
latitude
longitude
cycle
time
sea_level_anomaly
distance_from_theoretical
dac
ib_lf
internal_tide
ocean_tide
load_tide
validation_flag
mdt
swh
wind_speed
inter_mission_bias
distance_from_coast
```

To get attributs of the variable such as units, scale factor or offset, we can do :

In [78]:
```
f.variables['sea_level_anomaly']
```

Out[78]:
```
<class 'netCDF4._netCDF4.Variable'>
float32 sea_level_anomaly(nbpoints, nbcycles)
    _FillValue: 9.96921e+36
    long_name: Sea level anomaly with dac, ocean_tide, load_tide, interna
l_tide correction applied
    standard_name: sea_surface_height_above_sea_level
    units: m
    comment: The sea level anomaly is the sea surface height above mean s
ea surface height (MSS); It is computed with the following formula: sea_l
evel_anomaly = Orbit -Range - int inter_mission_bias - dac - ocean_tide -
load_tide - internal_tide - Solid_earth_tide - Pole_tide - Ionosphere - D
ry_troposphre - Wet_troposphere - SSB - MSS. Part of the corrections appl
ied are given in the following variables : inter_mission_biais, ocean_tid
e, load_tide, internal_tide, dac. See the product user manual for details
    quality_flag: validation_flag
    add_offset: 0.0
    scale_factor: 1.0
unlimited dimensions:
current shape = (21036, 199)
filling on
```

We can then use these attributs by doing this :

In [79]:
```
sla_units = f.variables['sea_level_anomaly'].units
sla_units
```

Out[79]:
```
'm'
```

In [80]:
```
lat_units = f.variables['latitude_theoretical'].units
lat_units
```

Out[80]:
```
'degrees_north'
```

The variable is in a masked array form (the mask corresponds to the presence of fill values) :

In [81]:
```
f.variables['sea_level_anomaly'][:]
```

Out[81]:
```
masked_array(
  data=[[--, --, --, ..., --, --, --],
        [--, 0.09000000357627869, 0.023000000044703484, ..., --, --, --],
        [--, 0.026000000536441803, 0.20499999821186066, ..., --,
         -0.3140000104904175, --],
        ...,
        [--, --, 6.480000019073486, ..., --, 4.826000213623047, --],
        [--, 12.888999938964844, 7.61899995803833, ..., --,
         5.783999919891357, --],
        [--, 10.314000129699707, 8.836000442504883, ..., --,
         5.98199987411499, --]],
  mask=[[ True,  True,  True, ...,  True,  True,  True],
        [ True, False, False, ...,  True,  True,  True],
        [ True, False, False, ...,  True, False,  True],
        ...,
        [ True,  True, False, ...,  True, False,  True],
        [ True, False, False, ...,  True, False,  True],
        [ True, False, False, ...,  True, False,  True]],
  fill_value=9.969209968386869e+36)
```

To access the variable and replace the fill values by NaN, we can do :

In [82]:
```
sla_raw = np.ma.filled(f.variables['sea_level_anomaly'][:], np.nan)
sla_raw
```

Out[82]:
```
array([[        nan,         nan,         nan, ...,         nan,
                nan,         nan],
       [        nan,  0.09      ,  0.023     , ...,         nan,
                nan,         nan],
       [        nan,  0.026     ,  0.205     , ...,         nan,
        -0.31400001,         nan],
       ...,
       [        nan,         nan,  6.48000002, ...,         nan,
         4.82600021,         nan],
       [        nan, 12.88899994,  7.61899996, ...,         nan,
         5.78399992,         nan],
       [        nan, 10.31400013,  8.83600044, ...,         nan,
         5.98199987,         nan]])
```

The scale factor and the offset are applied automatically when doing this. We can thus import the latitude and longitude, and visualize the track on the map :

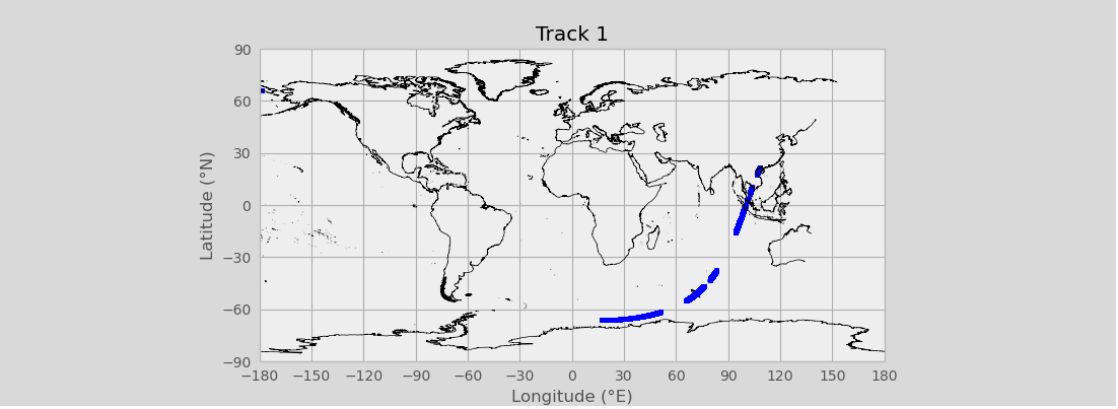
In [83]:
```
lat = np.ma.filled(f.variables['latitude_theoretical'][:], np.nan)
lat
```

Out[83]:
```
array([       nan, -66.147567, -66.147556, ...,  66.140121,  66.140186,
       66.140249])
```

In [85]:
```
lon = np.ma.filled(f.variables['longitude_theoretical'][:], np.nan)
lon
```

Out[85]:
```
array([  12.516352,   17.087164,   17.094015, ..., -178.522561,
       -178.515711, -178.50886 ])
```

In [86]:
```
projection_crs = ccrs.PlateCarree()
fig = plt.figure(figsize=(11,4))
ax = plt.axes(projection=projection_crs)
ax.coastlines(resolution='10m')
plt.scatter(lon, lat, c= 'b', marker='.', alpha=0.7)
ax.set_xticks(range(-180, 181, 30), crs=projection_crs)
ax.set_yticks(range(-90, 91, 30), crs=projection_crs)
ax.set_xlabel('Longitude (°E)')
ax.set_ylabel('Latitude (°N)')
plt.title('Track 1')
```



Out[86]:
```
Text(0.5, 1.0, 'Track 1')
```

There is a validation flag :

In [87]:
```
f.variables['validation_flag']
```

Out[87]:
```
<class 'netCDF4._netCDF4.Variable'>
int16 validation_flag(nbpoints, nbcycles)
    _FillValue: -32767
    long_name: Validation flag
    untis: 1
    meaning: 0 = valid measurement; 1 = invalid measurement
    values: 0, 1
unlimited dimensions:
current shape = (21036, 199)
filling on
```

In [90]:
```
flag = f.variables['validation_flag'][:]
flag
```

Out[90]:
```
masked_array(
  data=[[--, --, --, ..., --, --, --],
        [0, 0, 0, ..., 1, 1, 1],
        [0, 0, 0, ..., 1, 1, 1],
        ...,
        [1, 1, 1, ..., 1, 1, 1],
        [1, 1, 1, ..., 1, 1, 1],
        [--, 1, 1, ..., 1, 1, --]],
  mask=[[ True,  True,  True, ...,  True,  True,  True],
        [False, False, False, ..., False, False, False],
        [False, False, False, ..., False, False, False],
        ...,
        [False, False, False, ..., False, False, False],
        [False, False, False, ..., False, False, False],
        [ True, False, False, ..., False, False,  True]],
  fill_value=-32767,
  dtype=int16)
```

To better visualize the SLA and the validation flag, we can put it in DataFrame. The columns correspond to the cycles.

In [91]:
```
df_slaraw = pd.DataFrame(sla_raw)
df_slaraw
```

Out[91]:

	0	1	2	3	4	5	6	7	8	9	...	1
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	Na
1	NaN	0.090	0.023	0.060	0.142	-0.090	NaN	0.138000	-0.033	-0.079	...	0.1(
2	NaN	0.026	0.205	-0.033	0.189	0.192	NaN	0.163000	-0.111	0.059	...	0.0
3	NaN	0.027	0.081	0.077	0.035	0.048	0.035	0.054000	-0.028	-0.045	...	0.0
4	NaN	0.068	0.126	0.102	0.055	-0.101	0.143	0.043000	-0.087	0.124	...	0.0
...	...	...	...	...	...	...	...	...	...	...	...	
21031	NaN	6.184	NaN	5.622	7.127	5.590	5.103	-24.905001	NaN	NaN	...	5.5
21032	NaN	4.977	4.545	4.667	5.452	9.771	-2.804	32.222000	0.926	NaN	...	5.9
21033	NaN	NaN	6.480	9.332	7.194	11.338	12.356	3.564000	2.006	NaN	...	5.1
21034	NaN	12.889	7.619	12.073	8.925	11.667	8.091	11.469000	1.948	NaN	...	5.0
21035	NaN	10.314	8.836	13.663	10.985	9.567	9.978	10.425000	NaN	NaN	...	6.4

21036 rows × 199 columns

In [92]:
```
df_flag = pd.DataFrame(flag.data)
df_flag
```

Out[92]:

	0	1	2	3	4	5	6	7	8	9	...	
0	-32767	-32767	-32767	-32767	-32767	-32767	-32767	-32767	-32767	-32767	...	-3
1	0	0	0	0	0	0	-32767	0	0	0	...	
2	0	0	0	0	0	0	-32767	0	0	0	...	
3	0	0	0	0	0	0	0	0	0	0	...	
4	0	0	0	0	0	0	0	0	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
21031	1	1	1	1	1	1	1	1	1	1	...	
21032	1	1	1	1	1	1	1	1	1	1	...	
21033	1	1	1	1	1	1	1	1	1	1	...	
21034	1	1	1	1	1	1	1	1	1	1	...	
21035	-32767	1	1	1	1	1	1	1	-32767	1	...	

21036 rows × 199 columns

We create a boolean dataframe (acts like a mask) based on the validation flag. Here, the '0' (valid values) become the 'True'.

In [93]:
```
df_flag_bool = df_flag == 0
df_flag_bool
```

Out[93]:

	0	1	2	3	4	5	6	7	8	9	...	189	190	1
0	False	False	False	False	False	False	False	False	False	False	...	False	False	Fa
1	True	True	True	True	True	True	False	True	True	True	...	True	True	Tı
2	True	True	True	True	True	True	False	True	True	True	...	True	True	Tı
3	True	True	True	True	True	True	True	True	True	True	...	True	True	Tı
4	True	True	True	True	True	True	True	True	True	True	...	True	True	Tı
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
21031	False	False	False	False	False	False	False	False	False	False	...	False	False	Fa
21032	False	False	False	False	False	False	False	False	False	False	...	False	False	Fa
21033	False	False	False	False	False	False	False	False	False	False	...	False	False	Fa
21034	False	False	False	False	False	False	False	False	False	False	...	False	False	Fa
21035	False	False	False	False	False	False	False	False	False	False	...	False	False	Fa

21036 rows × 199 columns

To apply the validation flag on the SLA, we can do :

In [94]:
```
df_slaedit = df_slaraw[df_flag_bool]
df_slaedit
```

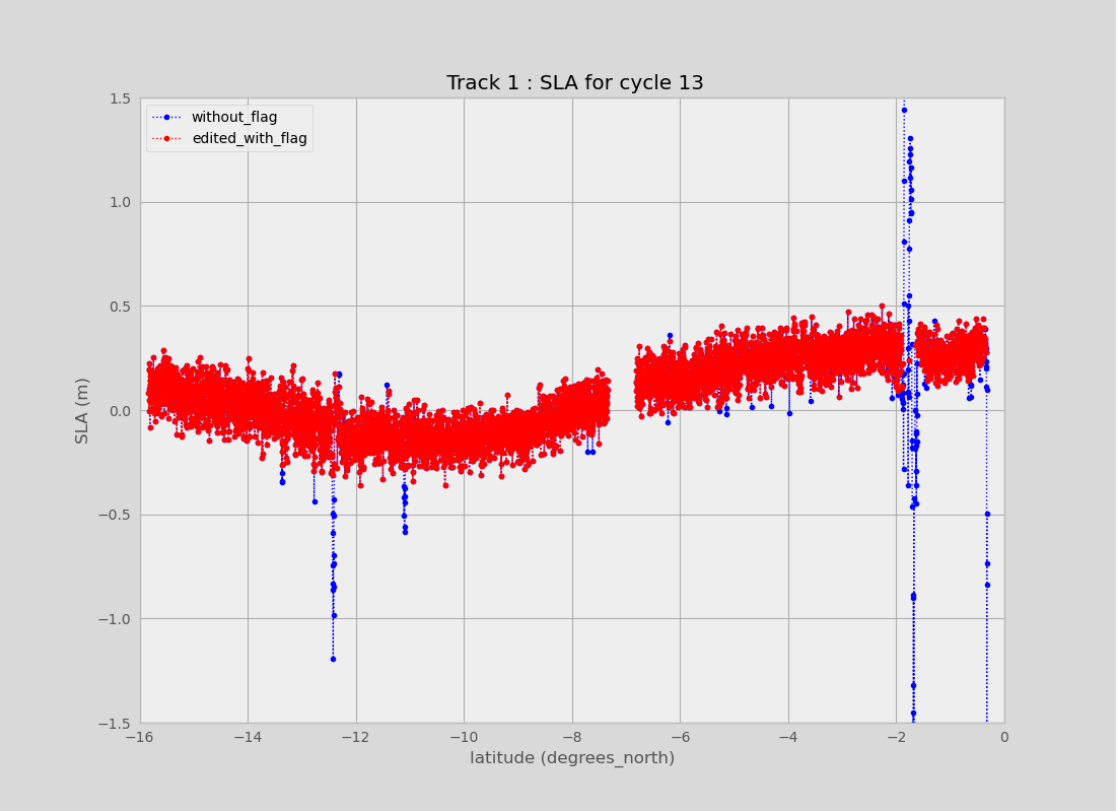
Out[94]:

	0	1	2	3	4	5	6	7	8	9	...	189	190
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
1	NaN	0.090	0.023	0.060	0.142	-0.090	NaN	0.138	-0.033	-0.079	...	0.100	0.167
2	NaN	0.026	0.205	-0.033	0.189	0.192	NaN	0.163	-0.111	0.059	...	0.023	0.054
3	NaN	0.027	0.081	0.077	0.035	0.048	0.035	0.054	-0.028	-0.045	...	0.025	0.132
4	NaN	0.068	0.126	0.102	0.055	-0.101	0.143	0.043	-0.087	0.124	...	0.022	0.074
...	...	...	...	...	...	...	...	...	...	...	...	...	...
21031	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
21032	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
21033	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
21034	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
21035	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN

21036 rows × 199 columns

If we want to plot the SLA versus the latitude for cycle 13, we can do :

In [95]:
```
plt.figure()
plt.plot(lat, df_slaraw[13],'b:.', label='without_flag')
plt.plot(lat, df_slaedit[13],'r:.', label='edited_with_flag')
plt.xlabel('latitude (' + lat_units + ')')
plt.ylabel('SLA (' + sla_units + ')')
plt.xlim([-16, 0])
plt.ylim([-1.5, 1.5])
plt.title(' Track 1 : SLA for cycle 13')
plt.legend()
plt.show()
```



We can see that there are some outliers in the raw data, that are eliminated in the edited SLA.

Time manipulation

In [96]:
```
time = np.ma.filled(f.variables['time'][:], np.nan)
df_time = pd.DataFrame(time)
df_time
```

Out[96]:

	0	1	2	3	4	5
0	24153.436641	24163.352286	24173.267946	24183.183604	24193.099259	24203.014905
1	24153.436641	24163.352287	24173.267947	24183.183605	24193.099259	24203.014906
2	24153.436642	24163.352287	24173.267947	24183.183606	24193.099260	24203.014907
3	24153.436643	24163.352288	24173.267948	24183.183607	24193.099260	24203.014907
4	24153.436643	24163.352289	24173.267949	24183.183607	24193.099261	24203.014908
...	...	...	...	...	...	...
21031	24153.475545	24163.391191	24173.306850	24183.222509	24193.138163	24203.053810
21032	24153.475547	24163.391191	24173.306851	24183.222510	24193.138164	24203.053811
21033	24153.475547	24163.391192	24173.306852	24183.222510	24193.138164	24203.053812
21034	24153.475548	24163.391192	24173.306852	24183.222511	24193.138165	24203.053812
21035	24153.475548	24163.391193	24173.306853	24183.222511	24193.138165	24203.053813

21036 rows × 199 columns

In [97]:
```
f.variables['time']
```

Out[97]:
```
<class 'netCDF4._netCDF4.Variable'>
float64 time(nbpoints, nbcycles)
    _FillValue: 9.969209968386869e+36
    standard_name: time
    long_name: Time of measurement in UTC
    units: days since 1950-01-01T00:00:00+00:00
    calendar: gregorian
unlimited dimensions:
current shape = (21036, 199)
filling on
```

The time values are not in the usual datetime format (they correspond to the days since 01/01/1950). To convert them into datetime, we can do :

In [98]:
```
tref = datetime.datetime(1950,1,1)
df_time = df_time.transpose()
for i in range(len(time)):
    df_time[i] = np.array([tref + datetime.timedelta(days=t) for t in tim
df_time = df_time.transpose()
df_time
```

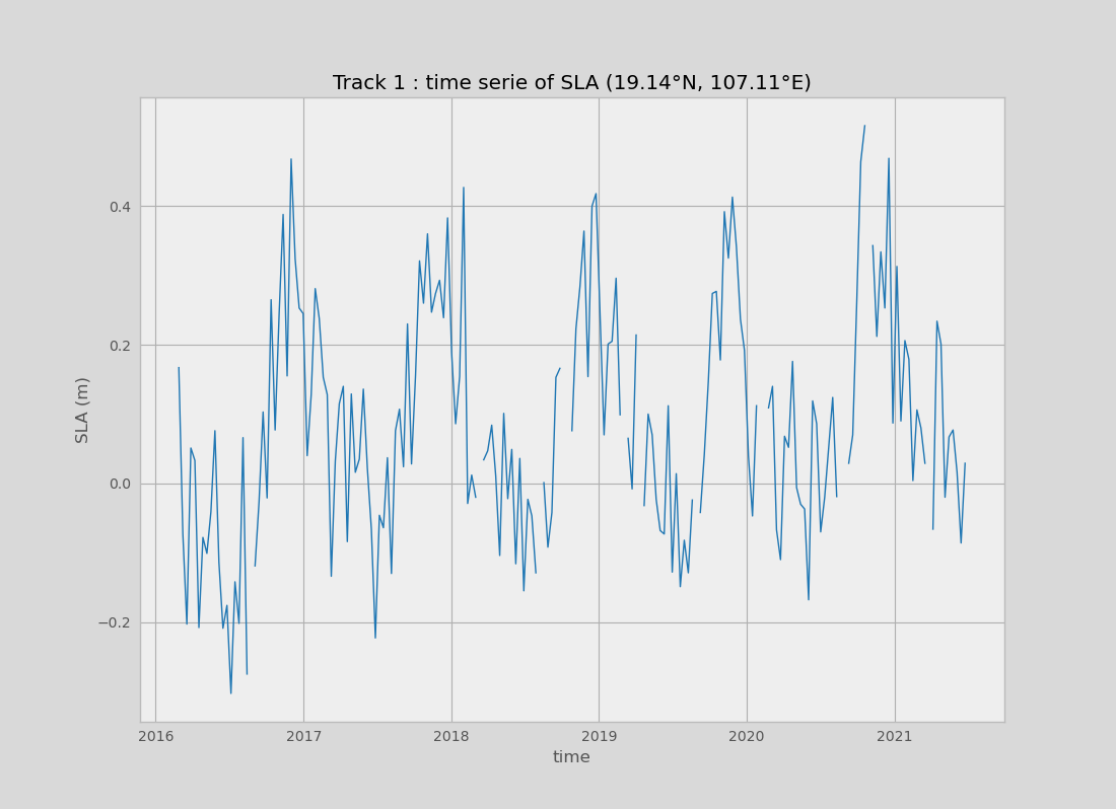
Out[98]:

	0	1	2	3	4	
0	2016-02-17 10:28:45.770699	2016-02-27 08:27:17.523518	2016-03-08 06:25:50.558704	2016-03-18 04:24:23.427726	2016-03-28 02:22:55.951534	00:
1	2016-02-17 10:28:45.821635	2016-02-27 08:27:17.574454	2016-03-08 06:25:50.609639	2016-03-18 04:24:23.478661	2016-03-28 02:22:56.002470	00:
2	2016-02-17 10:28:45.872571	2016-02-27 08:27:17.625390	2016-03-08 06:25:50.660574	2016-03-18 04:24:23.529596	2016-03-28 02:22:56.053406	00:
3	2016-02-17 10:28:45.923507	2016-02-27 08:27:17.676325	2016-03-08 06:25:50.711510	2016-03-18 04:24:23.631467	2016-03-28 02:22:56.104341	00:
4	2016-02-17 10:28:45.974442	2016-02-27 08:27:17.727261	2016-03-08 06:25:50.762445	2016-03-18 04:24:23.682403	2016-03-28 02:22:56.155277	00:
...	...	...	...	...	...	
21031	2016-02-17 11:24:47.119294	2016-02-27 09:23:18.868228	2016-03-08 07:21:51.868486	2016-03-18 05:20:24.787952	2016-03-28 03:18:57.281681	01:
21032	2016-02-17 11:24:47.221165	2016-02-27 09:23:18.919163	2016-03-08 07:21:51.919421	2016-03-18 05:20:24.838887	2016-03-28 03:18:57.332617	01:
21033	2016-02-17 11:24:47.272101	2016-02-27 09:23:18.970099	2016-03-08 07:21:51.970356	2016-03-18 05:20:24.889823	2016-03-28 03:18:57.383552	01:
21034	2016-02-17 11:24:47.323036	2016-02-27 09:23:19.021034	2016-03-08 07:21:52.021291	2016-03-18 05:20:24.940758	2016-03-28 03:18:57.434488	01:
21035	2016-02-17 11:24:47.373971	2016-02-27 09:23:19.071970	2016-03-08 07:21:52.072227	2016-03-18 05:20:24.991695	2016-03-28 03:18:57.485423	01:

21036 rows × 199 columns

We can then plot the time series of SLA at a given point :

In [99]:
```
lat_time_serie = np.round(lat[20000], 2)
lon_time_serie = np.round(lon[20000], 2)
plt.figure()
plt.plot(df_time.loc[20000], df_slaedit.loc[20000])
plt.xlabel('time')
plt.ylabel('SLA (' + sla_units + ')')
plt.title('Track 1 : time serie of SLA (' + str(lat_time_serie) + '°N, '
plt.show()
```

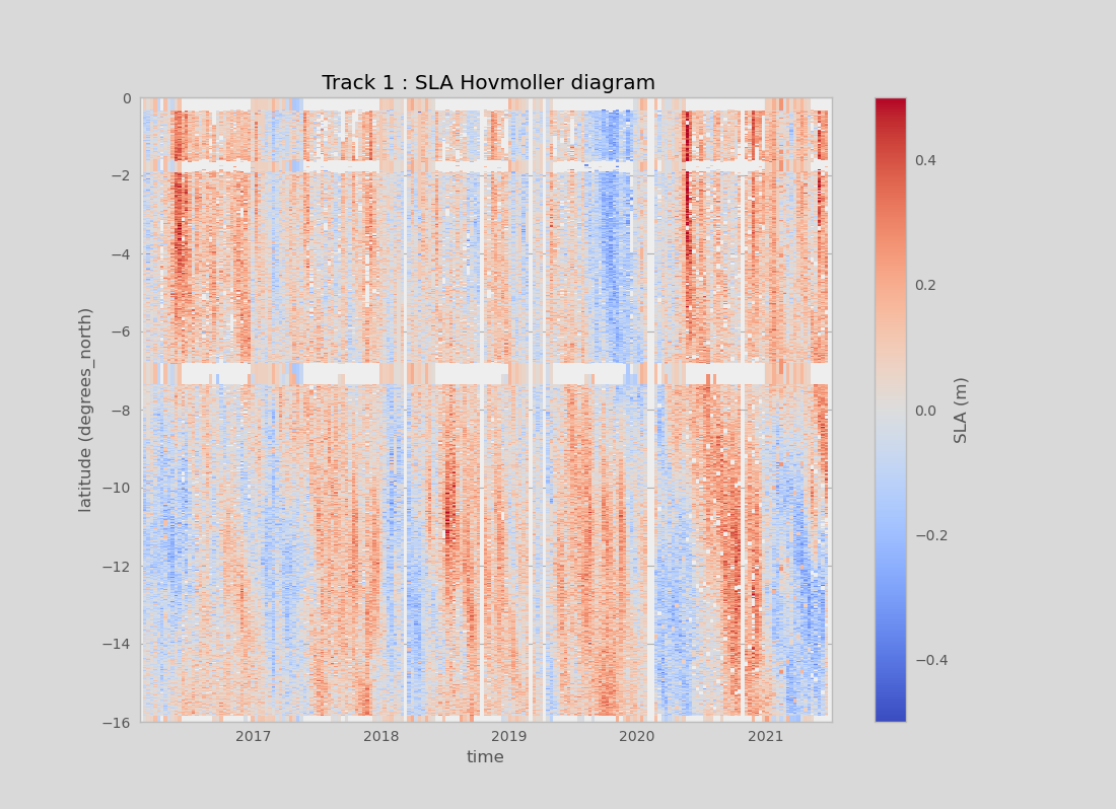


We can also plot Hovmoller diagrams of SLA (latitude vs time, with the SLA in color). For that, we need to replace the nans in the latitude by an outlier (that will not appear on the plot).

In [100…
```
lat = np.nan_to_num(lat, nan=100)
lat
```

Out[100]:
```
array([100.      , -66.147567, -66.147556, ...,  66.140121,  66.140186,
        66.140249])
```

In [101…
```
plt.figure()
plt.pcolormesh(df_time.values, lat, df_slaedit.values, vmin=-0.5, vmax=0.
plt.xlabel('time')
plt.ylabel('latitude (' + lat_units + ')')
plt.title(' Track 1 : SLA Hovmoller diagram')
plt.ylim([-16,0])
plt.colorbar(label='SLA (m)')
plt.show()
```



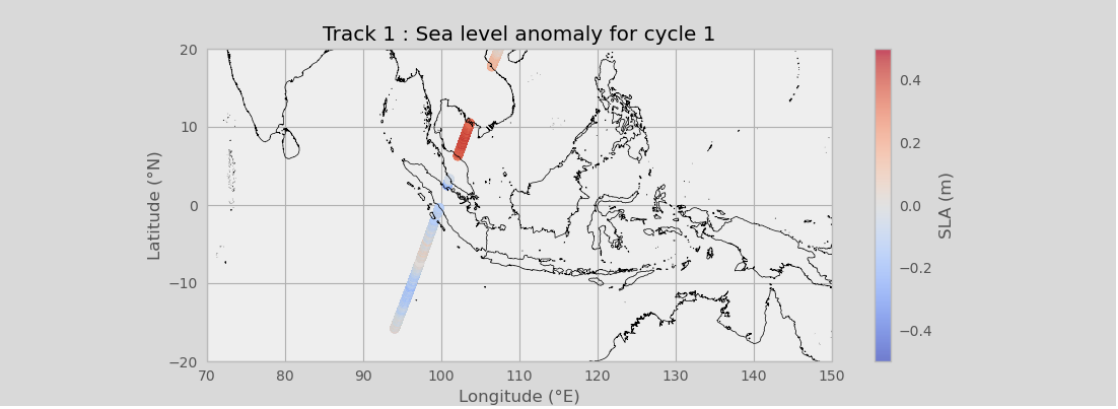
Maps

In [102…
```
lat = np.ma.filled(f.variables['latitude_theoretical'][:], np.nan)
lon = np.ma.filled(f.variables['longitude_theoretical'][:], np.nan)
```

In [1]:
```
6/4
```

Out[1]:
```
1.5
```

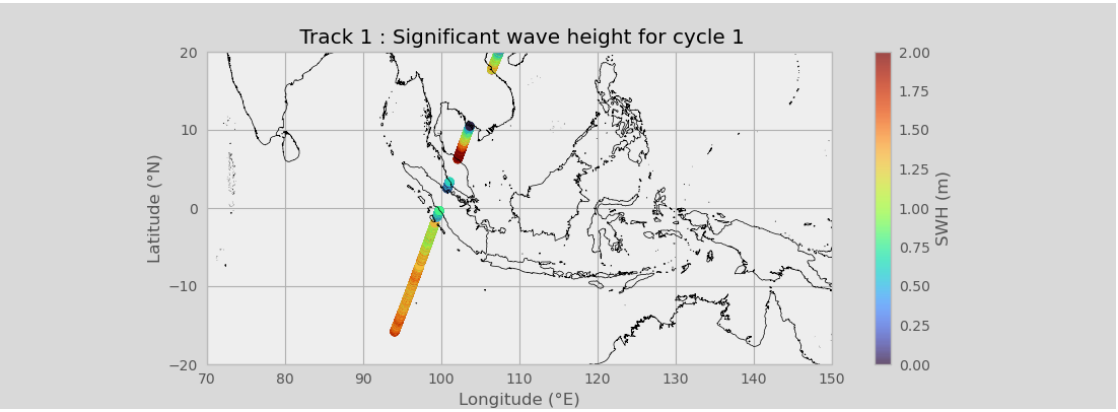
In [103…
```
projection_crs = ccrs.PlateCarree()
fig = plt.figure(figsize=(11,4))
ax = plt.axes(projection=projection_crs)
ax.coastlines(resolution='10m')
plt.scatter(lon, lat, c=df_slaedit[1], marker='o', alpha=0.7, vmin=-0.5,
ax.set_xlim([70, 150])
ax.set_ylim([-20, 20])
ax.set_xticks(range(70, 151, 10), crs=projection_crs)
ax.set_yticks(range(-20, 21, 10), crs=projection_crs)
ax.set_xlabel('Longitude (°E)')
ax.set_ylabel('Latitude (°N)')
ax.set_title('Track 1 : Sea level anomaly for cycle 1')
cb = plt.colorbar()
cb.set_label('SLA (m)')
```



We can also plot other variables present in the file :

In [104…
```
swh = np.ma.filled(f.variables['swh'][:], np.nan)
df_swhraw = pd.DataFrame(swh)
df_swhedit = df_swhraw[df_flag_bool]
wind = np.ma.filled(f.variables['wind_speed'][:], np.nan)
df_windraw = pd.DataFrame(wind)
df_windedit = df_windraw[df_flag_bool]
```

In [105…
```
projection_crs = ccrs.PlateCarree()
fig = plt.figure(figsize=(11,4))
ax = plt.axes(projection=projection_crs)
ax.coastlines(resolution='10m')
plt.scatter(lon, lat, c=df_swhedit[1], marker='o', alpha=0.7, vmin=0, vma
ax.set_xlim([70, 150])
ax.set_ylim([-20, 20])
ax.set_xticks(range(70, 151, 10), crs=projection_crs)
ax.set_yticks(range(-20, 21, 10), crs=projection_crs)
ax.set_xlabel('Longitude (°E)')
ax.set_ylabel('Latitude (°N)')
ax.set_title(' Track 1 : Significant wave height for cycle 1')
cb = plt.colorbar()
cb.set_label('SWH (m)')
```



In [106…

```
projection_crs = ccrs.PlateCarree()
fig = plt.figure(figsize=(11,4))
ax = plt.axes(projection=projection_crs)
ax.coastlines(resolution='10m')
plt.scatter(lon, lat, c=df_windedit[1], marker='o', alpha=0.7, vmin=0, vm
ax.set_xlim([70, 150])
ax.set_ylim([-20, 20])
ax.set_xticks(range(70, 151, 10), crs=projection_crs)
ax.set_yticks(range(-20, 21, 10), crs=projection_crs)
ax.set_xlabel('Longitude (°E)')
ax.set_ylabel('Latitude (°N)')
ax.set_title(' Track 1 : Wind speed for cycle 1')
cb = plt.colorbar()
cb.set_label('wind speed (m/s)')
```