CoastSat.slope: Narrabeen-Collaroy example

This is an extention of the main CoastSat toolbox and it is assumed that the user is familiar with CoastSat as the outputs of CoastSat are used here to estimate beach slopes. The `coastsat` environment also needs to be installed before attempting this example.

This example shows how to estimate the beach slope along 5 transects at Narrabeen-Collaroy, Sydney, Australia.

Initial settings

In [1]:

```
# initial settings
%load_ext autoreload
%autoreload 2
import os
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
import pytz
import pickle
# beach slope estmation module
import SDS_slope
```

1. Load satellite-derived shorelines and transect locations

Satellite-derived shorelines from Landsat 5, 7 and 8 between 1999 and 2020 are needed to estimate the beach slope, these have to be mapped with CoastSat beforehand. When mapping shorelines with CoastSat, the coordinates of the 2D shorelines are saved in a file named `sitename_output.pkl`.

In this example we use 2 files that are under *example_data/* (you will need the same files for another site):

- `NARRA_output.pkl` : satellite-derived shorelines mapped from 1999-2020 using Landsat 5,7 and 8 (no Sentinel-2)
- `NARRA_transects.geojson` : cross-shore transect coordinates (2 points, the first one being landwards)

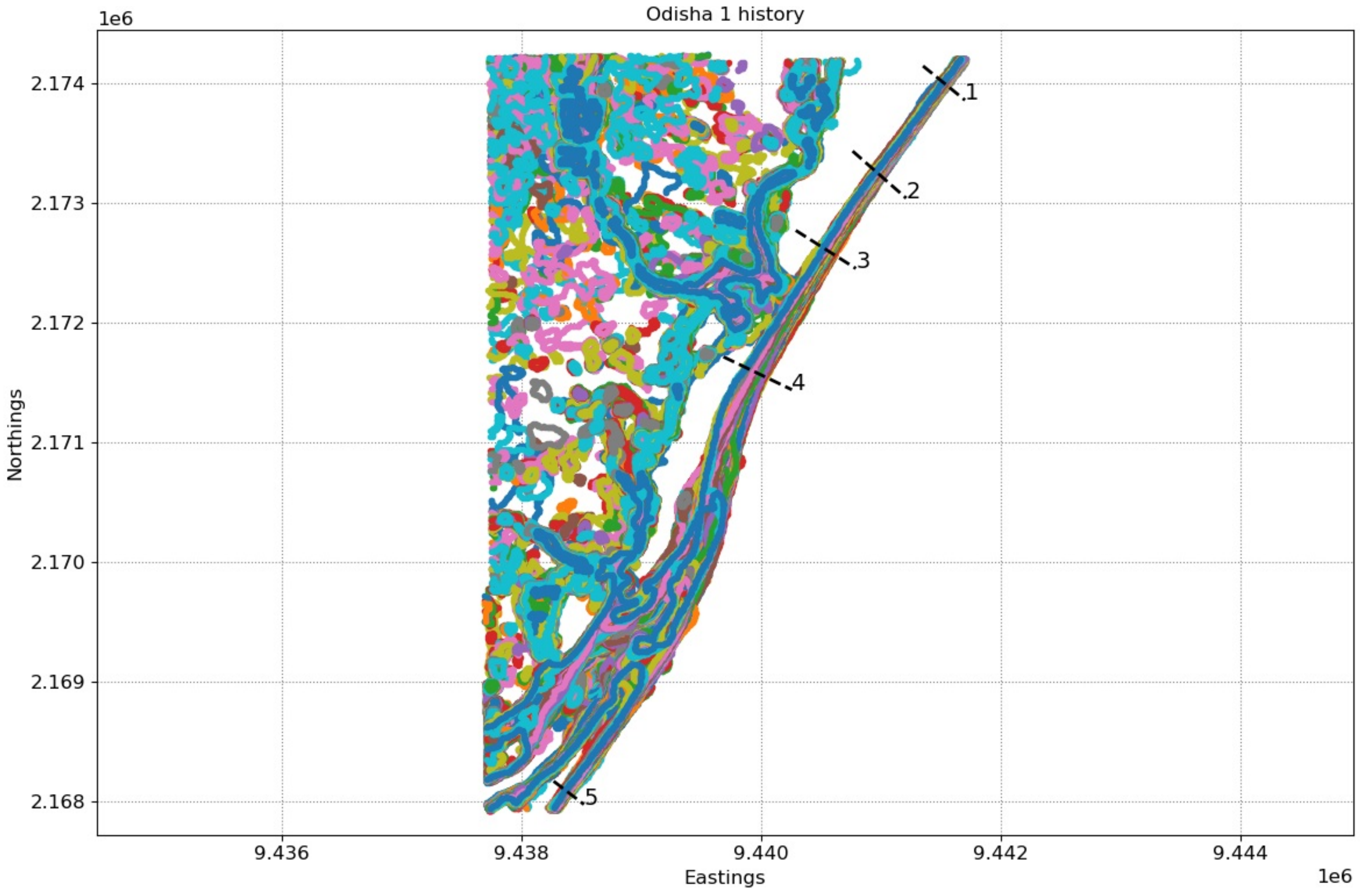
When preparing your own files, make sure that both files are in the same coordinate system (in this example epsg:28356).

The section below loads the two files, removes duplicates and shorelines with poor georeferencing and plots the 2D shorelines and cross-shore transects.

In [7]:

```
# load the sitename_output.pkl generated by CoastSat
sitename = 'Odisha 1 history'

with open(os.path.join(r'C:\Users\z5030440\OneDrive - UNSW\fes-2.9.1-Source\data\fes2014', sitename + '_output' +
    output = pickle.load(f)

# load the 2D transects from geojson file
geojson_file = os.path.join(os.getcwd(), r'C:\Users\z5030440\OneDrive - UNSW\fes-2.9.1-Source\data\fes2014', site
transects = SDS_slope.transects_from_geojson(geojson_file)

# remove S2 shorelines (the slope estimation algorithm needs only Landsat shorelines)
if 'S2' in output['satname']:
    idx_S2 = np.array([_ == 'S2' for _ in output['satname']])
    for key in output.keys():
        output[key] = [output[key][_] for _ in np.where(~idx_S2)[0]]

# remove duplicates (can happen that images overlap and there are 2 shorelines for the same date)
output = SDS_slope.remove_duplicates(output)
# remove shorelines from images with poor georeferencing (RMSE > 10 m)
output = SDS_slope.remove_inaccurate_georef(output, 10)

# plot shorelines and transects
fig,ax = plt.subplots(1,1,figsize=[12,  8])
fig.set_tight_layout(True)
ax.axis('equal')
ax.set(xlabel='Eastings', ylabel='Northings', title=sitename)
ax.grid(linestyle=':', color='0.5')
for i in range(len(output['shorelines'])):
    coords = output['shorelines'][i]
    date = output['dates'][i]
    ax.plot(coords[:,0], coords[:,1], '.', label=date.strftime('%d-%m-%Y'))
for key in transects.keys():
    ax.plot(transects[key][:,0],transects[key][:,1],'k--',lw=2)
    ax.text(transects[key][-1,0], transects[key][-1,1], key)
```

```
5 transects have been loaded
0 duplicates
8 bad georef
```



2. Extract time-series of shoreline change along the transects

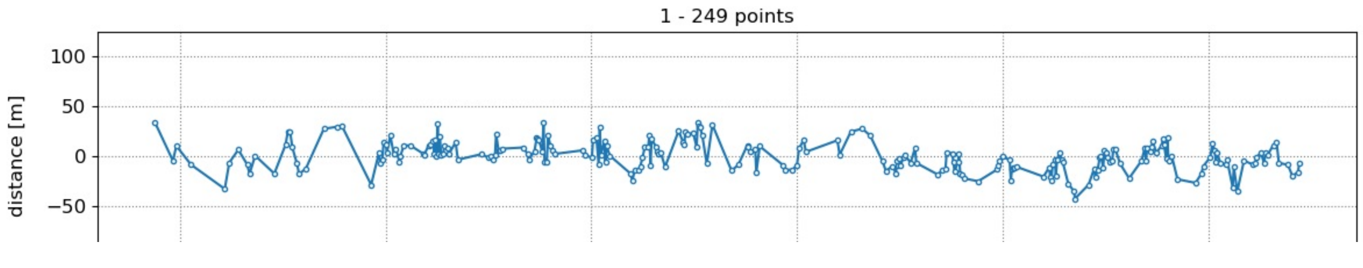
To obtain time-series of shoreline change we need to calculate the intersections between the 2D shorelines and the cross-shore transects, this can be done in the CoastSat toolbox but I provided here a more advanced method that deals with outliers and erroneous detections. As the accuracy of the beach slope estimate will depend on the quality of the satellite-derived shorelines, it is important to get rid of large outliers as these will affect the slope estimates.

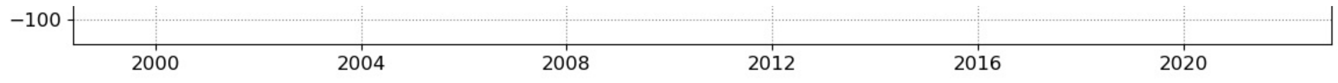
To remove outliers use the `max_cross_change` parameter to define the maximum cross-shore distance for despiking the time-series. Narrabeen-Collaroy is microtidal and storm-dominated, therefore the threshold was set at 40 m.

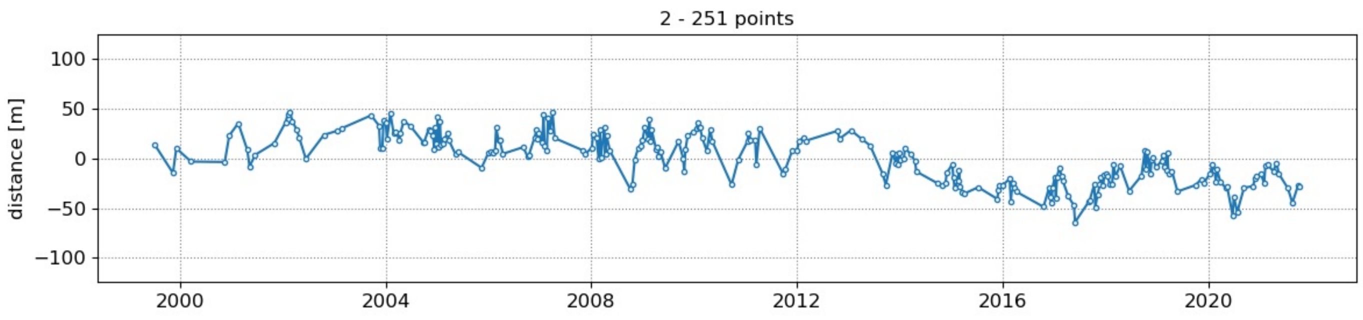
In [8]:

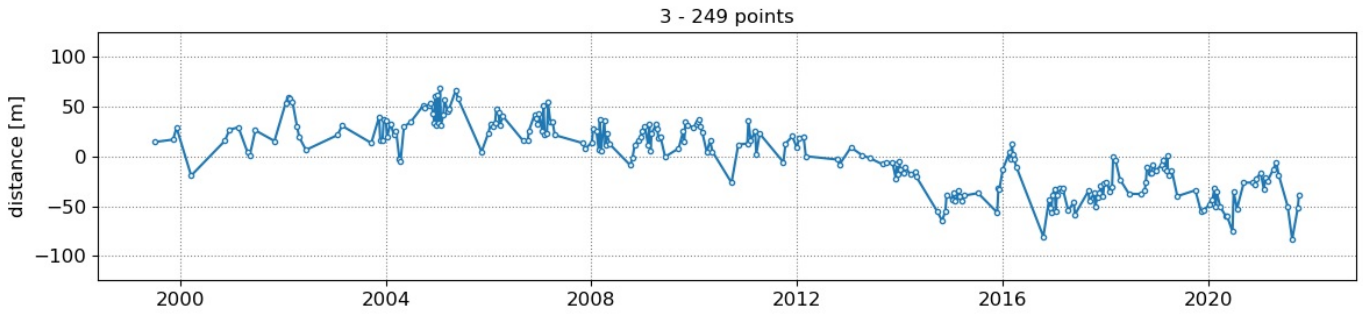
```
# a more robust method to compute intersections is provided here to avoid the presence of outliers in the time-se

settings_transects = {# parameters for shoreline intersections
                      'along_dist':         25,         # along-shore distance to use for intersection
                      'max_std':            15,         # max std for points around transect
                      'max_range':          30,         # max range for points around transect
                      'min_val':            -100,       # largest negative value along transect (landwards of tra
                      # parameters for outlier removal
                      'nan/max':            'auto',     # mode for removing outliers ('auto', 'nan', 'max')
                      'prc_std':            0.1,        # percentage to use in 'auto' mode to switch from 'nan' t
                      'max_cross_change':   40,         # maximum cross-shore distance for despiking.
                      }
# compute intersections [advanced version]
cross_distance = SDS_slope.compute_intersection(output, transects, settings_transects)
# remove outliers [advanced version]
cross_distance = SDS_slope.reject_outliers(cross_distance,output,settings_transects)
# plot time-series
SDS_slope.plot_cross_distance(output['dates'],cross_distance)
```

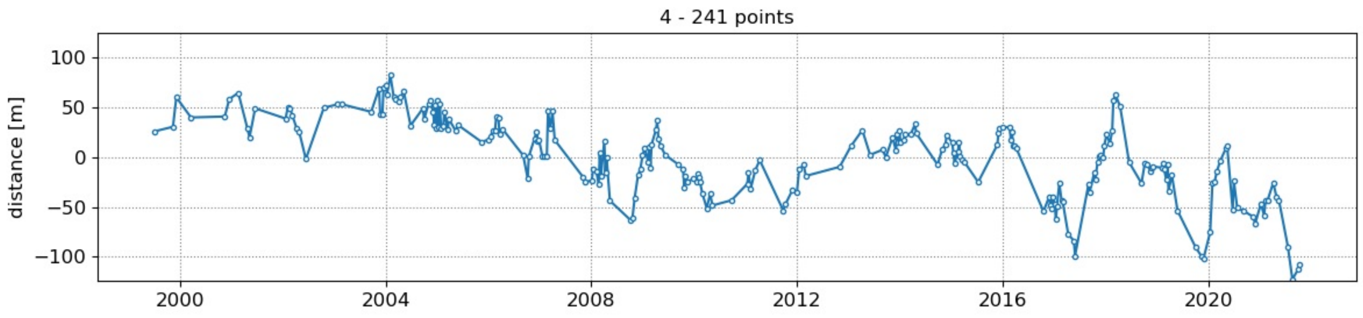
```
1  - outliers removed 1
2  - outliers removed 0
3  - outliers removed 1
4  - outliers removed 3
5  - outliers removed 1
```

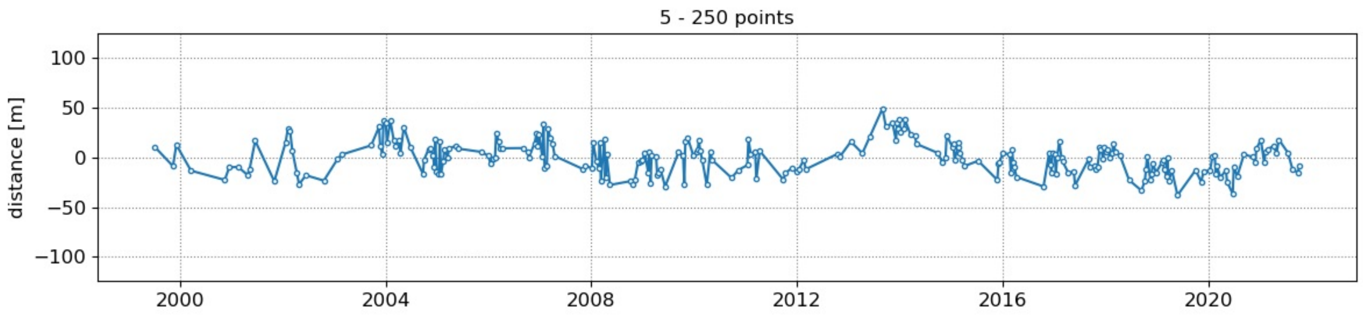












3. Get tide levels at the time of image acquisition

Now that we have the time-series of shoreline change, we need to obtain the tide level at the time of image acquisition for each data point. There are two options to get the tide levels:

- **Option 1:** Use a global tide model (FES2014 from AVISO) to get the modeled tide levels at the time of image acquisition
- **Option 2:** Provide your own file with measured/modeled tide levels

There are also some parameters to estimate the beach slope. You can change the trial beach slopes if the range does not correspond to the beach slope at your site by changing `slope_min` and `slope_max`. Do not change any of the other parameters.

In the section below the time-series of shoreline change are cropped between 1999 and 2000 as this is the period when 2 Landsat satellites are concurrently in orbit (providing a minimum sampling period of 8 days).

In [9]:

```
# slope estimation settings
days_in_year = 365.2425
seconds_in_day = 24*3600
settings_slope = {'slope_min':        0.035,                  # minimum slope to trial!
                  'slope_max':        1.0,                    # maximum slope to trial!
```

```
                   'delta_slope':      0.005,                  # slope increment
                   'date_range':       [1999,2020],            # range of dates over which to perform the analysis
                   'n_days':           8,                      # sampling period [days]
                   'n0':               50,                     # parameter for Nyquist criterium in Lomb-Scargle t
                   'freqs_cutoff':     1./(seconds_in_day*30), # 1 month frequency
                   'delta_f':          100*1e-10,              # deltaf for identifying peak tidal frequency band
                   'prc_conf':         0.05,                   # percentage above minimum to define confidence bar
                   }
settings_slope['date_range'] = [pytz.utc.localize(datetime(settings_slope['date_range'][0],5,1)),
                                pytz.utc.localize(datetime(settings_slope['date_range'][1],1,1))]
beach_slopes = SDS_slope.range_slopes(settings_slope['slope_min'], settings_slope['slope_max'], settings_slope['

# clip the dates between 1999 and 2020 as we need the Landsat satellites
idx_dates = [np.logical_and(_>settings_slope['date_range'][0],_<settings_slope['date_range'][1]) for _ in output[
dates_sat = [output['dates'][_] for _ in np.where(idx_dates)[0]]
for key in cross_distance.keys():
    cross_distance[key] = cross_distance[key][idx_dates]
```

In [10]:

```
# Storing all the variables I've got so far.
# At this juncture, I should:
  # 1) store variables to retrieve later
  # 2) interrupt Jupyter notebook, switch environments to pyfes envt, open the other notebook and restore variabl
%store sitename
%store output
%store geojson_file
%store transects
# %store fig
# %store ax
%store coords
%store date
%store cross_distance
%store settings_transects
%store days_in_year
%store seconds_in_day
%store settings_slope
%store beach_slopes
%store idx_dates
%store dates_sat

# At this pt, go and change to pyfes envt and continue
```

```
Stored 'sitename' (str)
Stored 'output' (dict)
Stored 'geojson_file' (str)
Stored 'transects' (dict)
Stored 'coords' (ndarray)
Stored 'date' (datetime)
Stored 'cross_distance' (dict)
Stored 'settings_transects' (dict)
Stored 'days_in_year' (float)
Stored 'seconds_in_day' (int)
Stored 'settings_slope' (dict)
Stored 'beach_slopes' (ndarray)
Stored 'idx_dates' (list)
Stored 'dates_sat' (list)
```

Option 1: get tide levels from FES2014

You will need to install FES2014 following the instructions provided here. Information about this global tide model can be found on AVISO's website.

In the section below the tide level corresponding to each date in `dates_sat` is computed from the model in a numpy.array named `tide_sat`.

In [ ]:

```
# Retrieved stored variables in pyfes envt environment (in other notebook)
# Don't run this cell here
%store -r sitename
%store -r output
%store -r geojson_file
%store -r transects
# %store -r fig
# %store -r ax
%store -r coords
%store -r date
%store -r cross_distance
%store -r settings_transects
%store -r days_in_year
%store -r seconds_in_day
%store -r settings_slope
%store -r beach_slopes
%store -r idx_dates
%store -r dates_sat
```

In [ ]:

```
# Precautions to avoid "DLL load failed":
    # Run Anaconda Prompt as admin
# Don't run this cell here
import os

# Option 1. if FES2014 global tide model is setup
import pyfes
# point to the folder where you downloaded the .nc files
filepath = r'C:\Users\z5030440\OneDrive - UNSW\fes-2.9.1-Source\data\fes2014'
config_ocean = os.path.join(filepath, 'ocean_tide.ini') # change to ocean_tide.ini
config_load =  os.path.join(filepath, 'load_tide.ini')  # change to load_tide.ini
ocean_tide = pyfes.Handler("ocean", "io", config_ocean)
load_tide = pyfes.Handler("radial", "io", config_load)
```

In [ ]:

```
# # Don't run this cell here
# Once again, store all the variables obtained so far
%store sitename
%store output
%store geojson_file
%store transects
%store fig
%store ax
%store coords
%store date
%store cross_distance
%store settings_transects
%store days_in_year
%store seconds_in_day
%store settings_slope
%store beach_slopes
%store idx_dates
%store dates_sat
%store filepath
%store config_ocean

%store config_load
%store ocean_tide
%store load_tide
%store coords
%store time_step
%store dates_fes
%store tide_fes
%store tide_sat
# %store fig
# %store ax
```

In [ ]:

```
# Don't run this cell here
# coordinates of the location (always select a point 1-2km offshore from the beach)
    # if the model returns NaNs, change the location of your point further offshore.
coords = [151.332209, -33.723772]
# get tide time-series with 15 minutes intervals
time_step = 15*60
dates_fes, tide_fes = SDS_slope.compute_tide(coords,settings_slope['date_range'],time_step,ocean_tide,load_tide)
# get tide level at time of image acquisition
tide_sat = SDS_slope.compute_tide_dates(coords, dates_sat, ocean_tide, load_tide)

# plot tide time-series
fig, ax = plt.subplots(1,1,figsize=(12,3), tight_layout=True)
ax.set_title('Sub-sampled tide levels')
ax.grid(which='major', linestyle=':', color='0.5')
ax.plot(dates_fes, tide_fes, '-', color='0.6')
ax.plot(dates_sat, tide_sat, '-o', color='k', ms=4, mfc='w',lw=1)
ax.set_ylabel('tide level [m]')
ax.set_ylim(SDS_slope.get_min_max(tide_fes));
```

In [2]:

```
# ONLY RUN THIS cell onwards after activating coastsat again and opening THIS notebook, retrieve stored variables
%store -r sitename
%store -r output
%store -r geojson_file
%store -r transects
# %store fig
# %store ax
%store -r coords
%store -r date
%store -r cross_distance
%store -r settings_transects
%store -r days_in_year
%store -r seconds_in_day
%store -r settings_slope
%store -r beach_slopes
%store -r idx_dates
%store -r dates_sat
```

```
%store -r filepath
%store -r config_ocean

%store -r config_load
%store -r coords
%store -r time_step
%store -r dates_fes
%store -r tide_fes
%store -r tide_sat
# %store fig
# %store ax
```

Option 2: load the tide levels from your own file

If you prefer to use measured water levels or astronomical tides from your own model, you can provide your own file with the tide levels associated with the dates at which the shorelines where mapped (`dates_sat`). An example is provided below, you will need to create a numpy.array called `tides_sat` which contains an array of tide levels corresponding to each date in `dates_sat` .

In [ ]:

```
# Option 2. load tide levels corresponding to "dates_sat" from a file
# with open(os.path.join('example_data', sitename + '_tide' + '.pkl'), 'rb') as f:
#     tide_data = pickle.load(f)
# tide_sat = tide_data['tide']
# print(tides_sat)
```

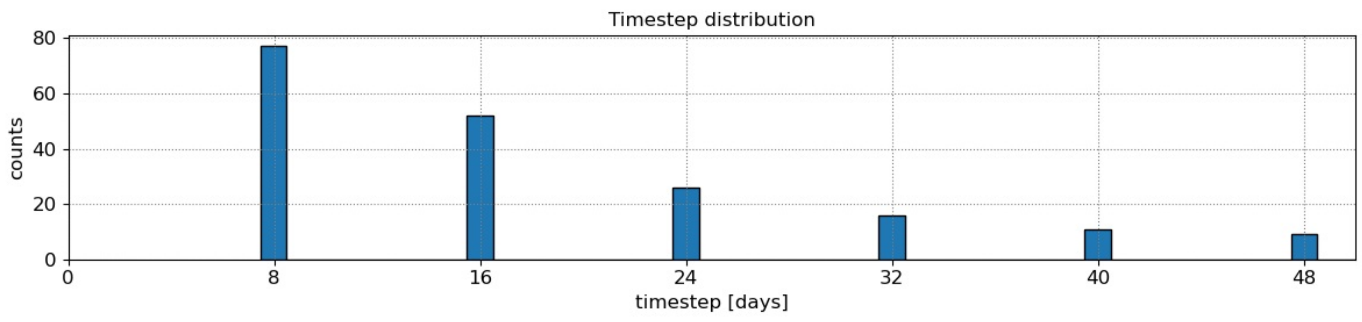
4. Peak tidal frequency

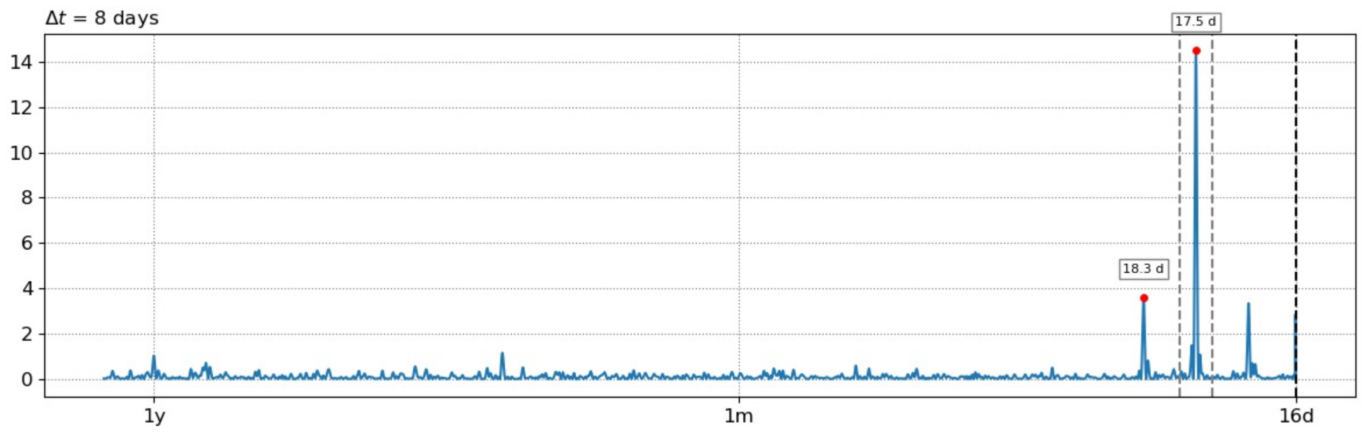
Find the peak tidal frequency, frequency band at which the energy is the largest in the subsampled tide level time-series.

Most sites will have a minimum sampling period of 8 days, but it can happen that because of overlapping images at some sites, a minimum sampling period of 7 days is achieved, then you can use 7 days instead of 8 by setting `settings_slope['n_days] = 7` . Don't use a sampling period of less than 7 days. If the plot of timestep distribution doesn't show a peak at 7 or 8 days, you will not be able to apply this technique as you don't have enough images.

In [5]:

```
# plot time-step distribution
t = np.array([_.timestamp() for _ in dates_sat]).astype('float64')
delta_t = np.diff(t)
fig, ax = plt.subplots(1,1,figsize=(12,3), tight_layout=True)
ax.grid(which='major', linestyle=':', color='0.5')
bins = np.arange(np.min(delta_t)/seconds_in_day, np.max(delta_t)/seconds_in_day+1,1)-0.5
ax.hist(delta_t/seconds_in_day, bins=bins, ec='k', width=1);
ax.set(xlabel='timestep [days]', ylabel='counts',
       xticks=settings_slope['n_days']*np.arange(0,20),
       xlim=[0,50], title='Timestep distribution');

# find tidal peak frequency
settings_slope['n_days'] = 8
settings_slope['freqs_max'] = SDS_slope.find_tide_peak(dates_sat,tide_sat,settings_slope)
```





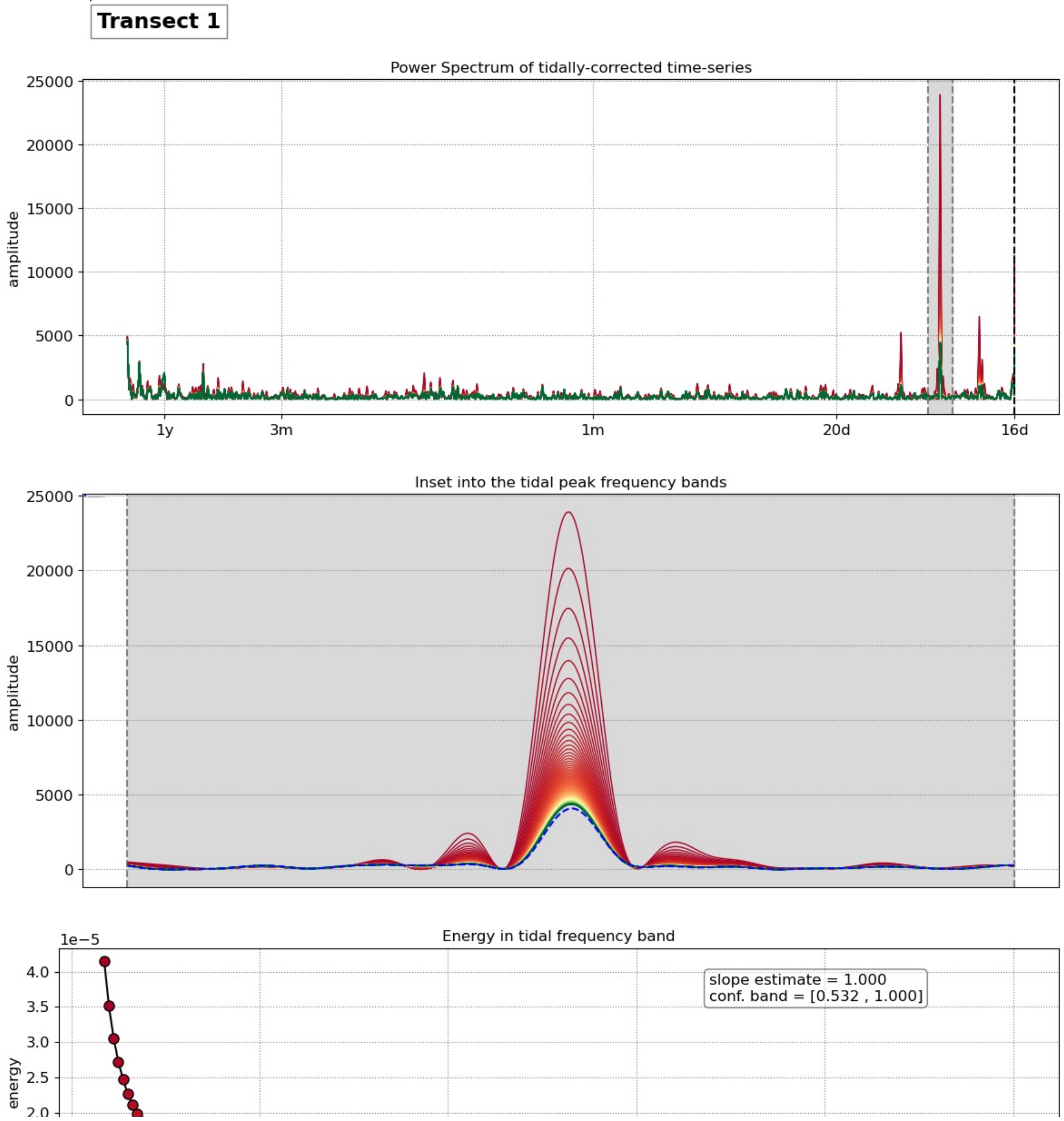
5. Estimate the beach slope

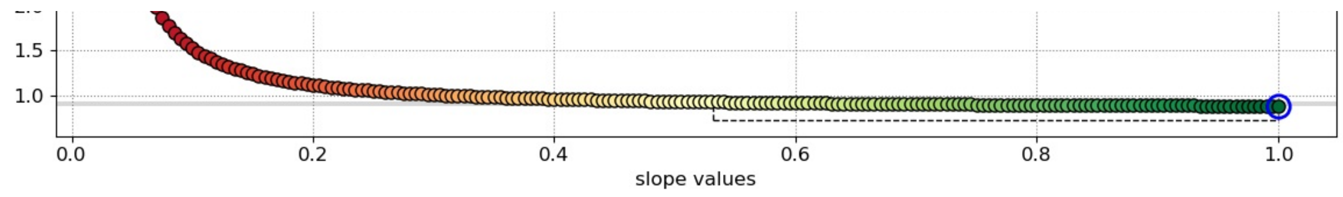
The beach slope along each transect is estimated by finding the slope that, when used for tidal correction, minimises the energy in the peak tidal frequency band. Based on our validation study, this slopes corresponds to the beach-face slope between mean sea level (MSL) and mean high water springs (MHWS).

In [6]:

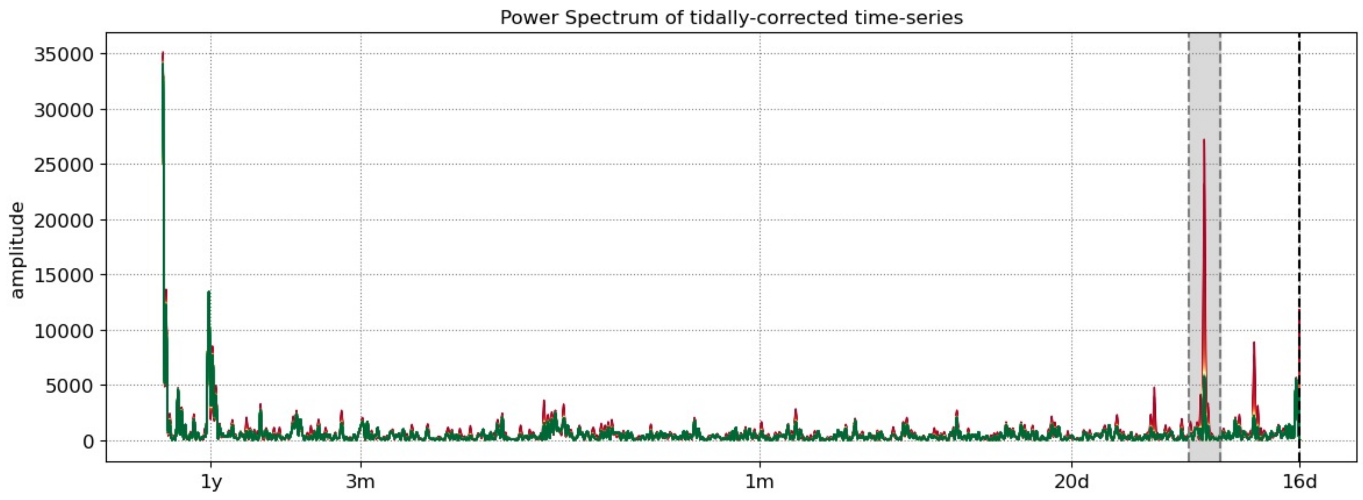
```
# estimate beach-face slopes along the transects
slope_est, cis = dict([]), dict([])
for key in cross_distance.keys():
    # remove NaNs
    idx_nan = np.isnan(cross_distance[key])
    dates = [dates_sat[_] for _ in np.where(~idx_nan)[0]]
    tide = tide_sat[~idx_nan]
    composite = cross_distance[key][~idx_nan]
    # apply tidal correction
    tsall = SDS_slope.tide_correct(composite,tide,beach_slopes)
    title = 'Transect %s'%key
    SDS_slope.plot_spectrum_all(dates,composite,tsall,settings_slope, title)
    slope_est[key],cis[key] = SDS_slope.integrate_power_spectrum(dates,tsall,settings_slope)
    print('Beach slope at transect %s: %.3f'%(key, slope_est[key]))
```

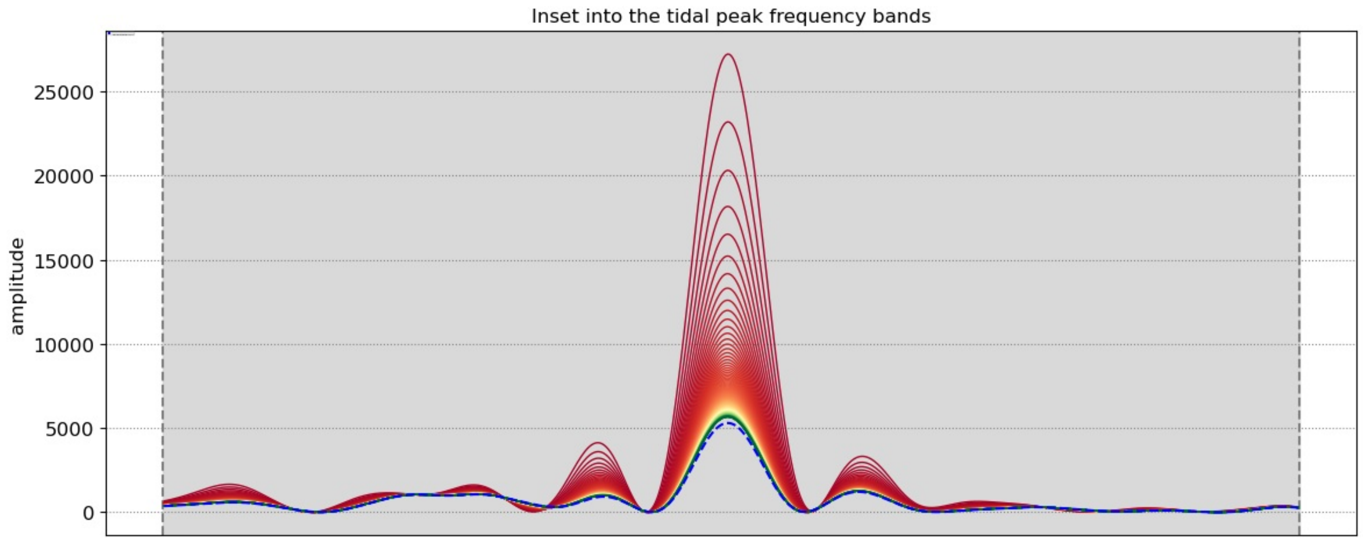
```
Beach slope at transect 1: 1.000
Beach slope at transect 2: 1.000
Beach slope at transect 3: 1.000
Beach slope at transect 4: 1.000
Beach slope at transect 5: 1.000
```

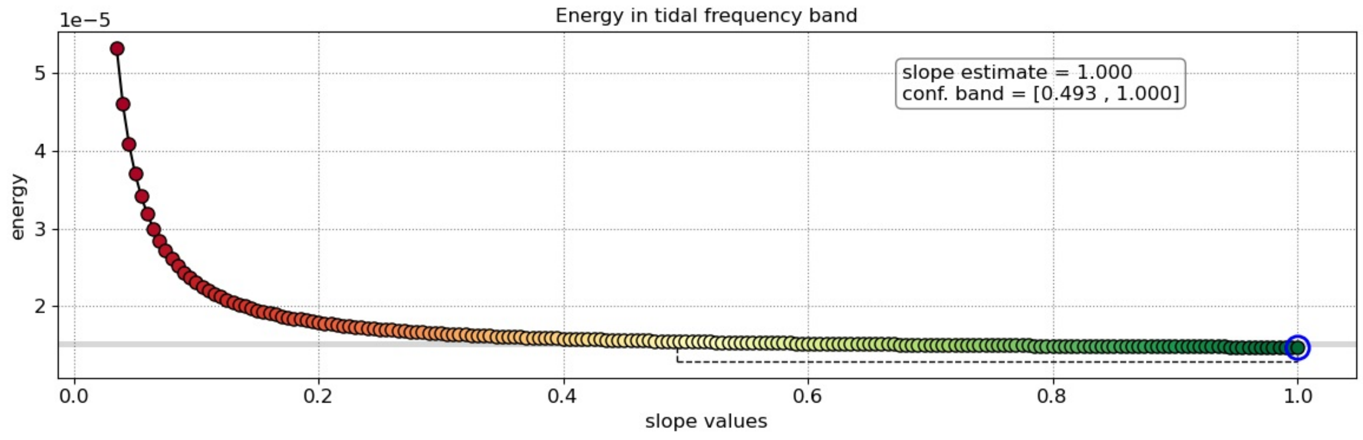




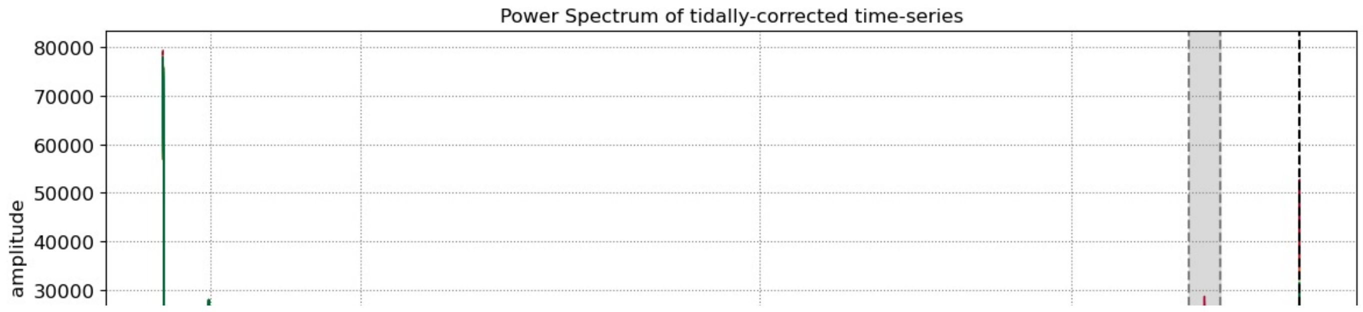
Transect 2

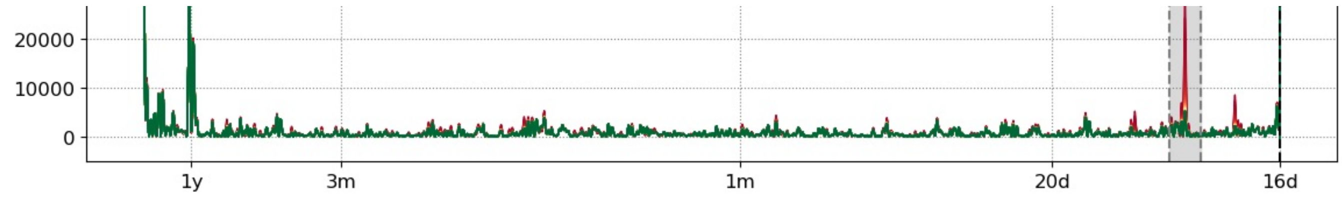


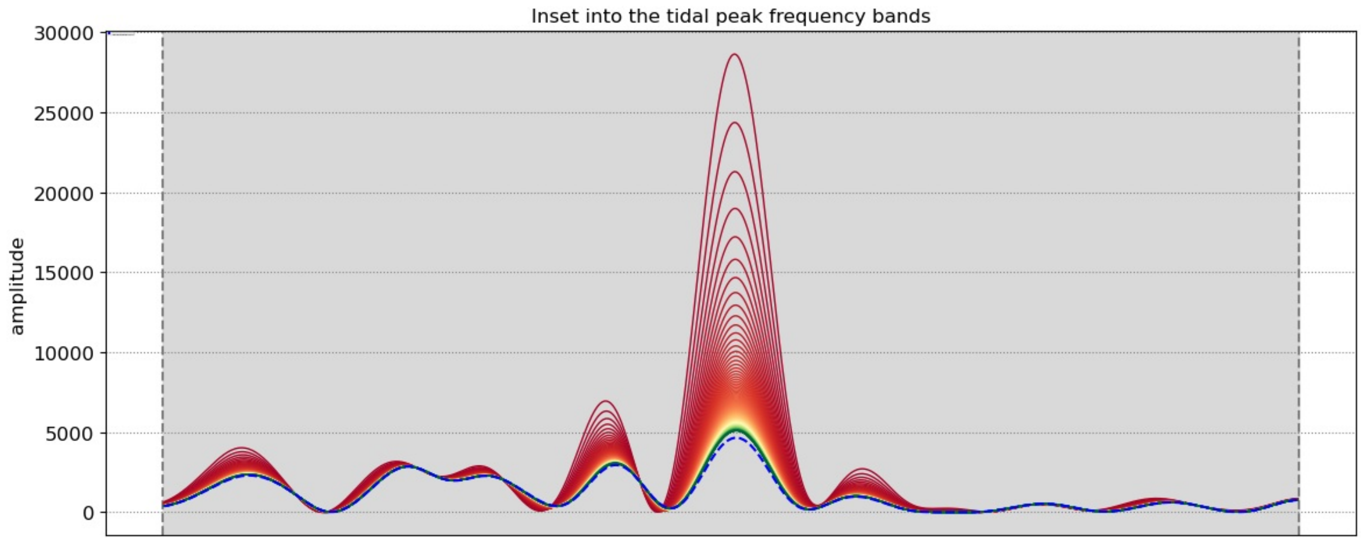


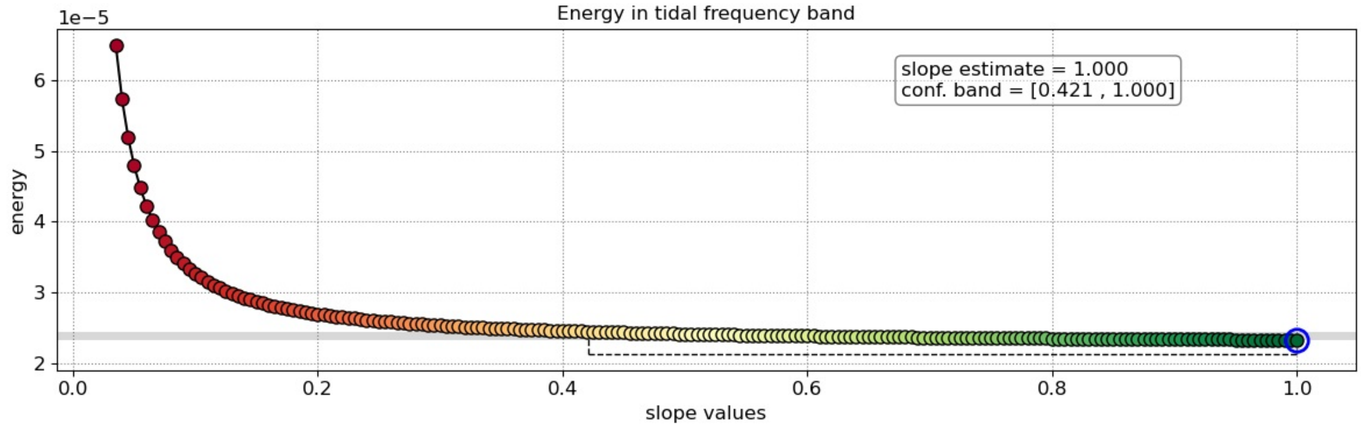


Transect 3

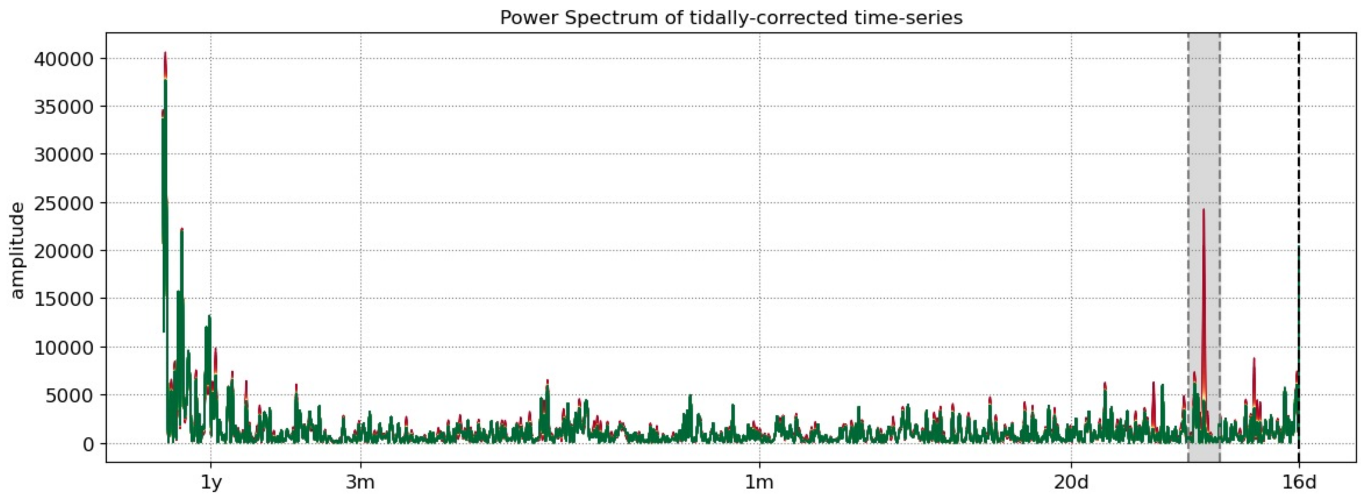


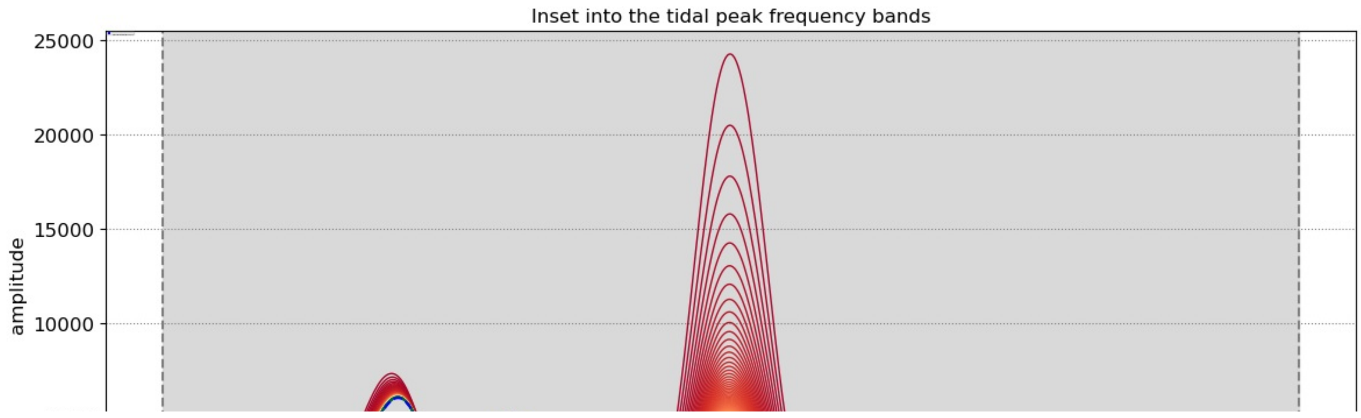

20000
10000
0
1y
3m
1m
20d
16d

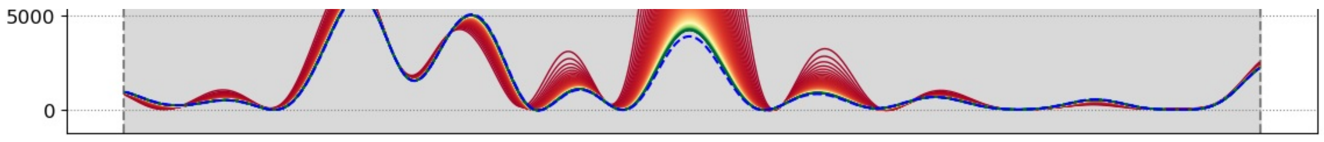

Inset into the tidal peak frequency bands
amplitude
30000
25000
20000
15000
10000
5000
0

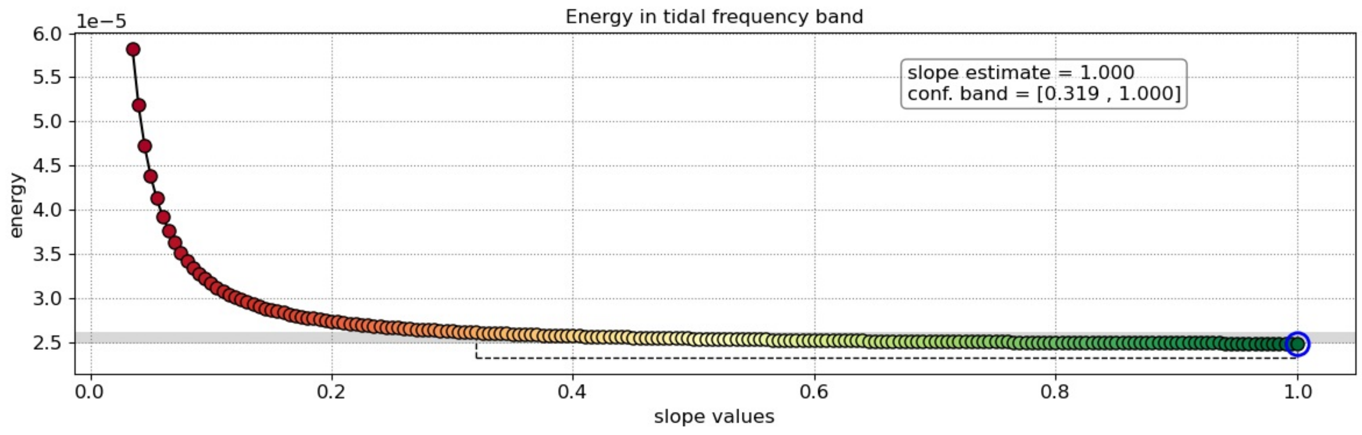

Energy in tidal frequency band
1e−5
energy
6
5
4
3
2
slope estimate = 1.000
conf. band = [0.421 , 1.000]
0.0
0.2
0.4
0.6
0.8
1.0
slope values

Transect 4

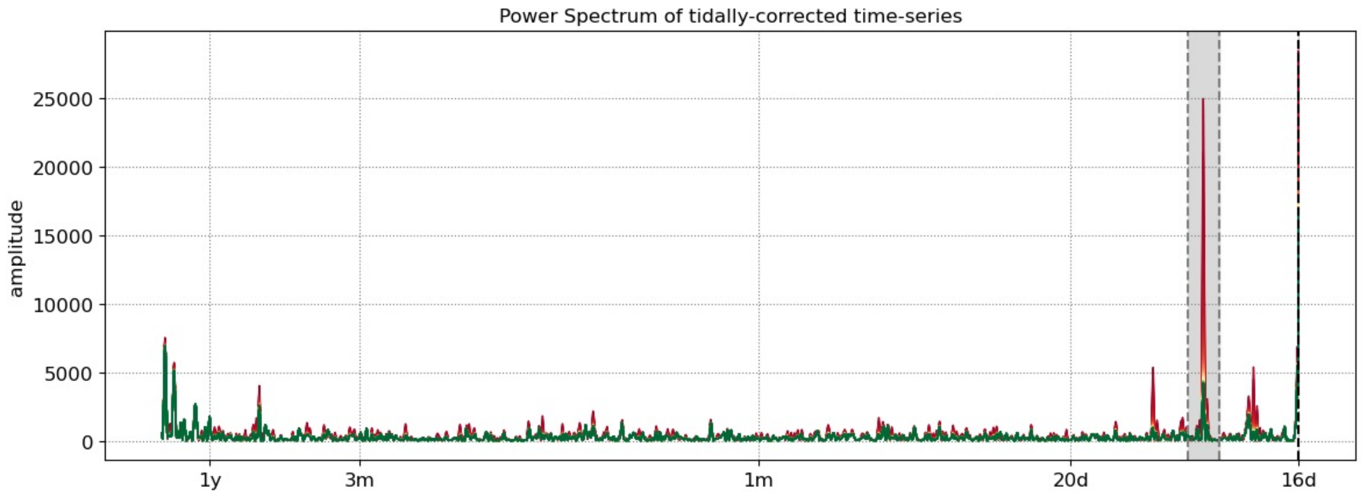

Power Spectrum of tidally-corrected time-series
amplitude
40000
35000
30000
25000
20000
15000
10000
5000
0
1y
3m
1m
20d
16d

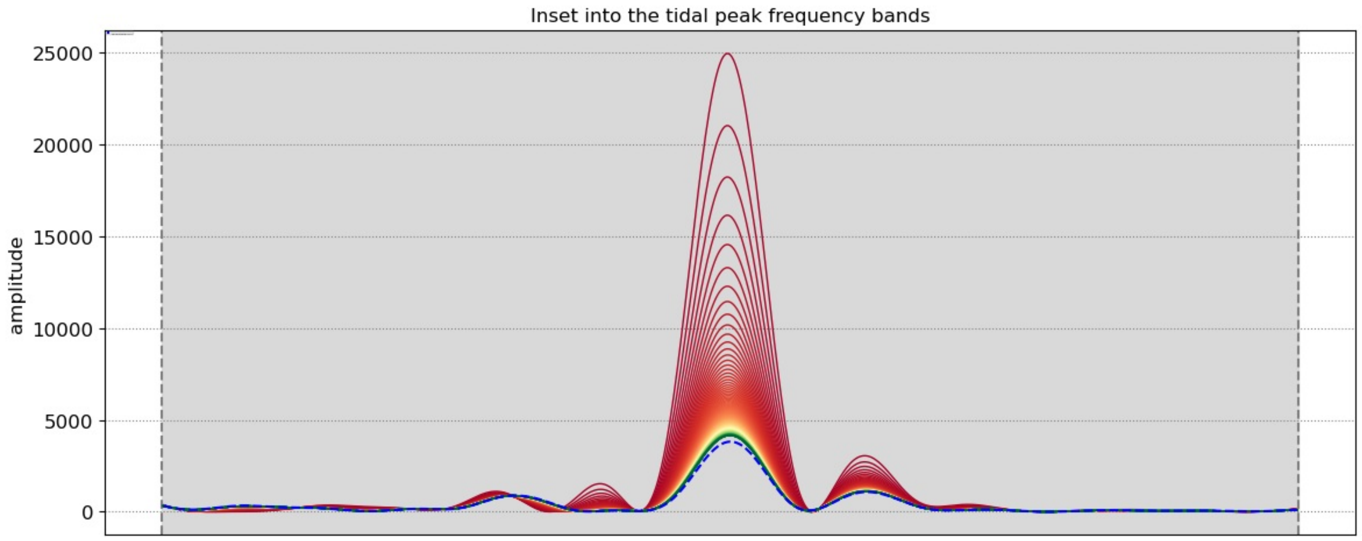

Inset into the tidal peak frequency bands
amplitude
25000
20000
15000
10000

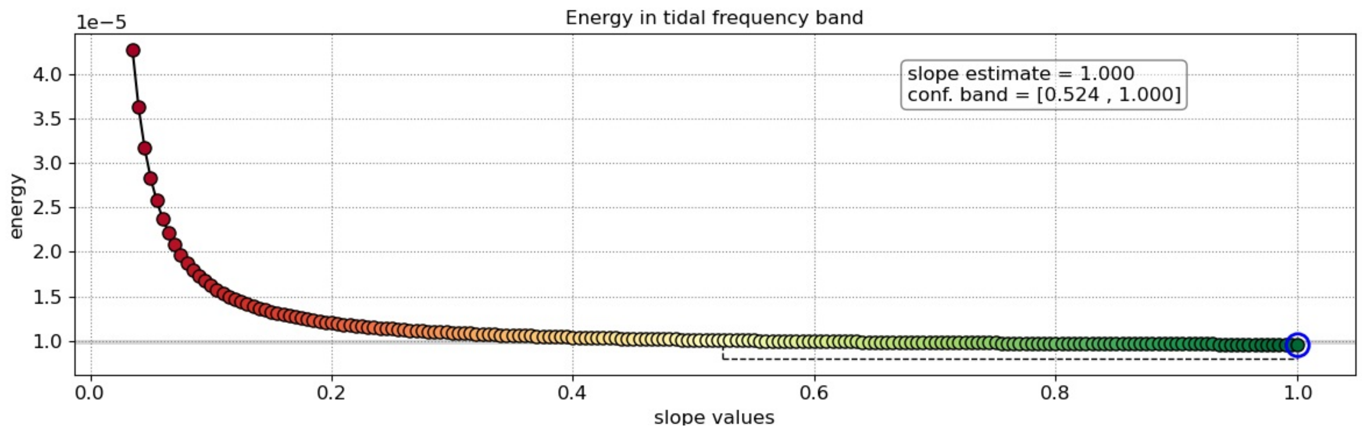

5000
0


Energy in tidal frequency band
1e−5
energy
slope estimate = 1.000
conf. band = [0.319 , 1.000]
slope values
0.0
0.2
0.4
0.6
0.8
1.0

Transect 5


Power Spectrum of tidally-corrected time-series
amplitude
1y
3m
1m
20d
16d


Inset into the tidal peak frequency bands
amplitude


Energy in tidal frequency band
1e−5
energy
slope estimate = 1.000
conf. band = [0.524 , 1.000]
slope values

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js