Mesh_generator

June 20, 2022

1 Create a mesh from a dataset

1.1 Import libraries and load dataset

[1]:

```
import matplotlib.pyplot as plt
import matplotlib.tri as tri
import numpy as np
import pandas as pd
```

1.2 Read data

[2]:

```
maritime_data = pd.read_csv("./Topography_arcachon/maritime_data.csv", sep=",",␣
 ↪header=2)
earth_data = pd.read_csv("./Topography_arcachon/earth_data copy.csv", sep=",",␣
 ↪header=3)
```

1.3 Data processing

Transform csv data to numpy arrays to be plotted

Correct for real maximum water depth (opposite of dataset depth)

[3]:

```
maritime_data.columns = ["long", "lat", "depth"]
maritime_data["depth"] = maritime_data["depth"]-1.9860
maritime_depth = maritime_data["depth"]
maritime_long = maritime_data["long"].to_numpy()
maritime_lat = maritime_data["lat"].to_numpy()

earth_data=pd.DataFrame(np.
 ↪array(earth_data),columns=['lat','long','depth','iteration'])
earth_depth = earth_data["depth"].to_numpy()
earth_long = earth_data["long"].to_numpy()
earth_lat = earth_data["lat"].to_numpy()
```

Changing scale

[4]:

```
maritime_depth = maritime_depth*30
maritime_long = maritime_long*111000
```

```
maritime_lat = maritime_lat*111000

earth_depth=earth_depth*30
earth_lat=earth_lat*111000
earth_long=earth_long*111000
```

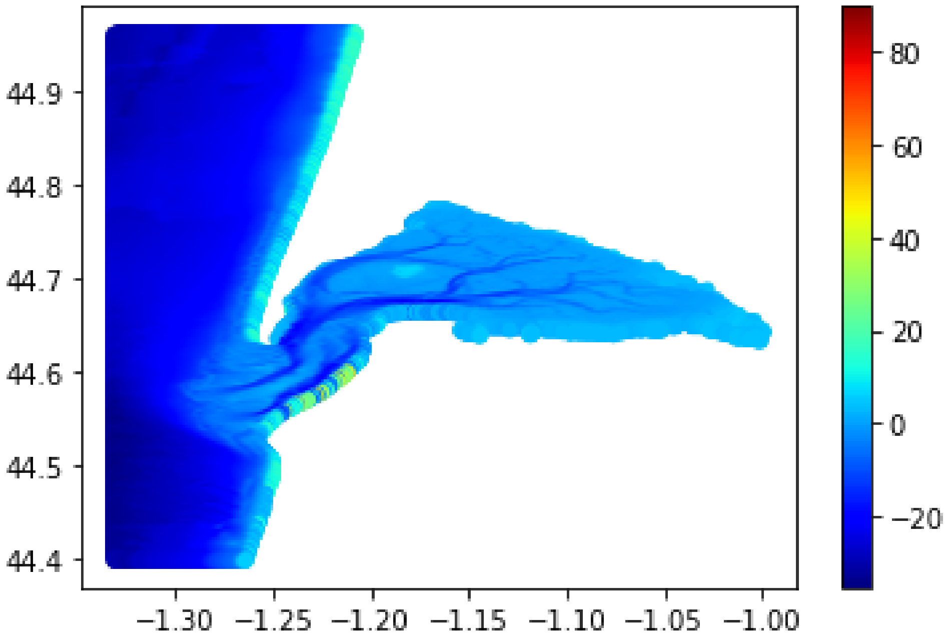
2 Plot datasets

3 Depth scale on the plots -> 1:1

Plot maritime data

[5]:

```
plt.figure()
plt.scatter(maritime_long/111000, maritime_lat/111000, c=maritime_depth/30,␣
 ↪cmap="jet",linewidths=0.0000005)
plt.clim(vmin=np.min(maritime_depth/30),vmax=np.max(earth_depth/30))
plt.colorbar()
plt.savefig("Images/Mar_data.png")
plt.show()
```

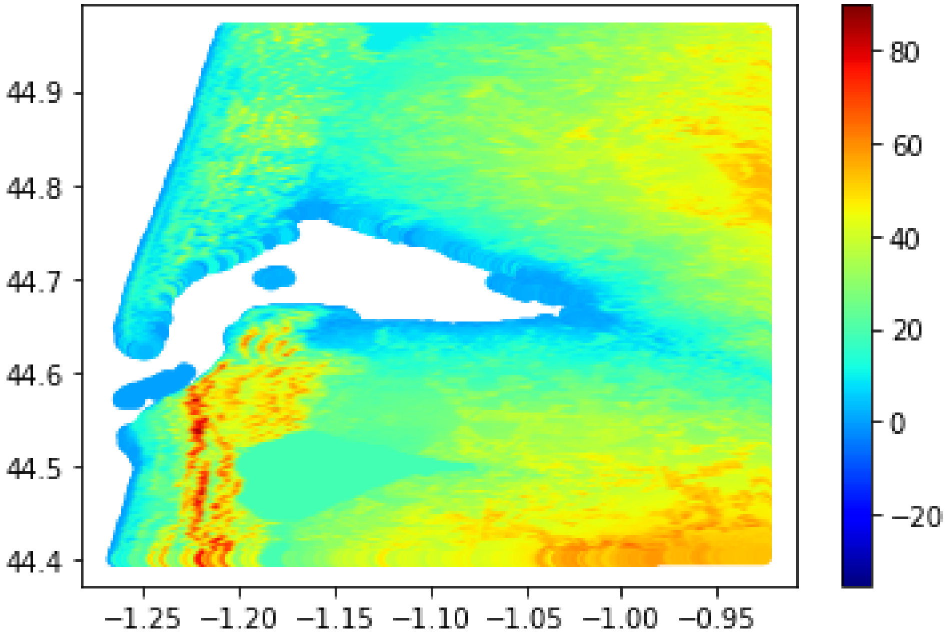


Plot topography data

[6]:

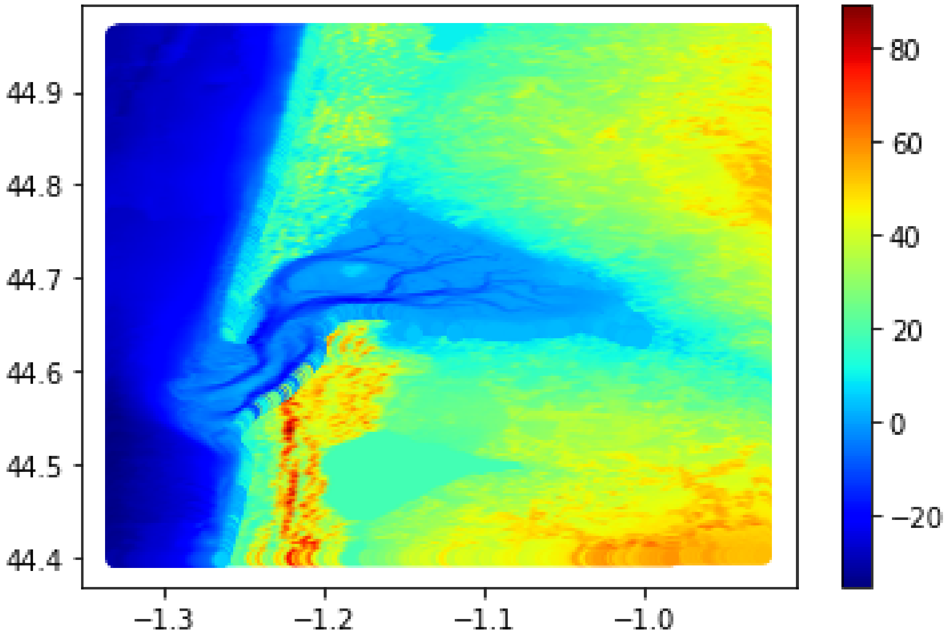
```
plt.figure()
plt.scatter(earth_long/111000, earth_lat/111000, c=earth_depth/30,␣
 ↪cmap="jet",linewidths=0.0000005)
```

```
plt.clim(vmin=np.min(maritime_depth)/30,vmax=np.max(earth_depth)/30)
plt.colorbar()
plt.savefig("Images/Topo_data.png")
plt.show()
```



Plot topography and maritime data

[7]:
```
plt.figure()
plt.scatter(np.concatenate((earth_long/111000,maritime_long/111000)), np.
 ↪concatenate((earth_lat/111000,maritime_lat/111000)), c=np.
 ↪concatenate((earth_depth/30,maritime_depth/30)), cmap="jet",linewidths=0.
 ↪0000005)
plt.clim(vmin=int(np.min(maritime_depth)/30),vmax=int(np.max(earth_depth)/30))
plt.colorbar()
plt.savefig("Images/Topo_Mar_data.png")
plt.show()
```



4 Select dataset

options=0 # 0: earth+maritime, 1: earth, other cases: maritime

[8]:

```
options=0 # 0: earth+maritime, 1: earth, other cases: maritime
if options==0:
    long=np.concatenate((maritime_long,earth_long))
    lat=np.concatenate((maritime_lat,earth_lat))
    depth=np.concatenate((maritime_depth,earth_depth))
elif options==1:
    long=earth_long
    lat=earth_lat
    depth=earth_depth
else :
    long=maritime_long
    lat=maritime_lat
    depth=maritime_depth
```

5 Utilisation of k-nearest neighbors algorithme and its average depth to make the dataset more smooth

[9]:

```
from sklearn.neighbors import NearestNeighbors

def smooth_grid(points,k_neigh=7):
    #points = points[points[:, 1].argsort()]
    #points = points[points[:, 0].argsort(kind='mergesort')]

    new_long=[]
    new_lat=[]
    new_depth=[]

    long=points[:,0]
    lat=points[:,1]
    matrix=np.array([long,lat]).T

    neigh = NearestNeighbors(n_neighbors=k_neigh)
    neigh.fit(matrix)
    NearestNeighbors(algorithm='auto', leaf_size=30)

    for i in range(len(matrix)):
        neighbors=neigh.kneighbors([[matrix[i,0],matrix[i,1]]])[1][0]
        average_long=(np.sum([points[k,0] for k in neighbors])+points[i,0])/
 ↪(k_neigh+1)
        average_lat=(np.sum([points[k,1] for k in neighbors])+points[i,1])/
 ↪(k_neigh+1)
        average_depth=(np.sum([points[k,2] for k in neighbors])+points[i,2])/
 ↪(k_neigh+1)
        new_long.append(average_long)
        new_lat.append(average_lat)
        new_depth.append(average_depth)

    return np.array([new_long,new_lat,new_depth]).T
```

[10]:

```
points = np.array([long,lat,depth]).T
points=smooth_grid(points,7)
```

5.1 Mesh creation

[11]:

```
import pyvista as pv
cloud = pv.PolyData(points)
grid = cloud.delaunay_2d(alpha=3000)   # We use the Delaunay method to make the␣
 ↪mesh
print("Number of points",grid.number_of_points)
print("Number of cells",grid.number_of_cells)
```

```
Number of points 138317
Number of cells 275469
```

6 Coloring cells and points

7 Depth scale -> 1:1

[12]:

```
import vtk
colors = vtk.vtkIntArray()
colors.SetName("Depth (m)")
depth_scale=[]
for i in range(grid.number_of_cells):
    cell=grid.cell_points(i)
    sum=0
    for j in range(len(cell)):
        sum+=cell[j][2]/30
    sum=int(sum/len(cell))
    depth_scale.append(sum)
for t in depth_scale:
    colors.InsertNextValue(t)
grid.GetCellData().SetScalars(colors)
```

[12]:

```
0
```

[13]:

```
colors = vtk.vtkIntArray()
colors.SetName("Depth (m)")
depth_scale=[]
for i in range(grid.number_of_points):
    depth_scale.append(int(points[i][2]/30))
for t in depth_scale:
    colors.InsertNextValue(t)
grid.GetPointData().SetScalars(colors)
```

[13]:

```
0
```

8 Plot and save to vtk file

[14]:

```
grid.plot(show_edges=True)
grid.save('mesh_arcachon.vtk')
```

```
ViewInteractiveWidget(height=768, layout=Layout(height='auto', width='100%'),␣
 ↪width=1024)
```