In [1]:

```
import os
from math import nan
import sys
import h5py
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import scipy
from scipy import signal
import numpy as np
import pandas as pd
import scipy.sparse as sparse
from mpl_toolkits.axes_grid1 import make_axes_locatable
sys.path.insert(0,"C:\CSES")
import map
from map import Map
```

In [ ]:

```
def MakeGraph(matrix_name, dataframe, vmax, vmin, binsize=0.5, tit=None, save=None ): #isLog=False
  bin_size_y = 0.5
  bin_size_x = 0.5

  lat_name = 'GEO_LAT'
  long_name = 'GEO_LON'
  bins, matrix = Map.generate_2d_histogram(dataframe.copy(), lat_name, long_name,matrix_name, bin_size_y, bin_size_x)
  if tit==None:
    tit = f"{matrix_name}_"+OrbitNumber+"_"+DATE[0]
# matrix[1] = media dei valori
# matrix[2] = denominatore
# matrix[0] = numeratore
  Map.show_map(bins[0], bins[1], matrix[1], tit, f"{file_name}/{tit}.jpg", vmax=vmax, vmin=vmin)
```

In [ ]:

```
def getData(data):

    Bw = 51200 * 2 / 1024
    data = pd.DataFrame(data)
    matrix=[]
    for i in range(len(data)):
      matrix.append(data.iloc[i])
    matrix=np.array(matrix)

    dataX = np.empty(shape=(matrix.shape[0], matrix.shape[1]))
    for i in range (0, matrix.shape[0]):
            meanX_b = np.mean(matrix[i])
            dataX[i] = matrix[i] - meanX_b


    M_b = dataX.shape[1]
    hamming_b = signal.get_window("hamming", M_b)
    FFT_low = np.array([scipy.fft.fft(dataX[i] * hamming_b) for i in range(0, dataX.shape[0])])
    out = np.abs(FFT_low.T[:1024]) ** 2
    outX_b = 400 + 20 * np.log10(out / Bw)


    return outX_b

def powerSpectrum(pow):

    powerX = 400 + 20 * np.log10(pow)

    return powerX
```

In [ ]:

```
def frequency (freq):
    sampleFreq = 51200
    nRow = 1024
    maxFreq = sampleFreq / 2
    freqRow = maxFreq / nRow
    row = int(freq / freqRow)
    return row



def Amplitude2(arr):
    mask = ~np.isnan(arr[row])
    dataX=arr[row][mask]
    for i in range(0, arr.shape[0]):
      if i!=row:
        mask = ~np.isnan(arr[i])
        arr[i][mask] = np.nan
    return arr
```

In [ ]:

```
dir_name = ""
file_name = dir_name + "C:/CSES/"
ext = ('.h5')
for path, dirc, files in os.walk(file_name):
    for name in files:
        if name.endswith('.h5'):
            OrbitNumber = name.split("_")[6]
            with h5py.File(str(file_name) + str(name), "r") as f:
                UTC_TIME = f["UTC_TIME"][()][:, 0]
                GEO_LAT = f["GEO_LAT"][()][:, 0]
                GEO_LON = f["GEO_LON"][()][:, 0]
                ALT = f["ALTITUDE"][()][:, 0]
                Workmode = f["WORKMODE"][()][:, 0]
                MAG_LAT = f["MAG_LAT"][()][:, 0]
                MAG_LON = f["MAG_LON"][()][:, 0]
                VERSE_TIME = f["VERSE_TIME"][()][:, 0]
                A131_W = f["A131_W"][()]
                A132_W = f["A132_W"][()]
                A133_W = f["A133_W"][()]
                A131_P = f["A131_P"][()]
                A132_P = f["A132_P"][()]
                A133_P = f["A133_P"][()]
                columns = list(f.keys())
                df = pd.DataFrame([])
                print(f)
                for column in columns:
                    try:
                        data = np.array(f[column])
                        if data.shape[1] == 1:
                            df[column] = data.flatten()
                        elif column.endswith('_P'):  # Getting only A131,A132,A133 _P data
                            mat = sparse.coo_matrix(data, shape=data.shape)
                            df[column] = mat.toarray().tolist()
                        elif column == "A131_W":
                            selected_data = np.array(data[0:len(Workmode), :])
                            mat = sparse.coo_matrix(selected_data, shape=selected_data.shape)
                            df[column] = mat.toarray().tolist()
                        elif column == "A132_W":
                            selected_data = np.array(data[0:len(Workmode), :])
                            mat = sparse.coo_matrix(selected_data, shape=selected_data.shape)
                            df[column] = mat.toarray().tolist()
                        elif column == "A133_W":
```

```
            selected_data = np.array(data[0:len(Workmode), :])
            mat = sparse.coo_matrix(selected_data, shape=selected_data.shape)
            df[column] = mat.toarray().tolist()
        else:
            print(column + ' skipped')
    except Exception as e:
        pass
S_burst = df[df.WORKMODE == 2]

df['DATE_TIME'] = pd.to_datetime(df.UTC_TIME, format='%Y%m%d%H%M%S%f')
DATE = df.DATE_TIME.map(lambda x: x.strftime('%Y-%m-%d'))
TIME = df.DATE_TIME.map(lambda x: x.strftime('%H-%M-%S'))
date_burst = pd.to_datetime(S_burst.UTC_TIME, format='%Y%m%d%H%M%S%f')
TIME_BURST = date_burst.map(lambda x: x.strftime('%H-%M-%S'))

powerX= powerSpectrum(A131_P)
powerY = powerSpectrum(A132_P)
powerZ = powerSpectrum(A133_P)



temp_df_x = []
for i in range(len(df)):
    if(df.iloc[i].WORKMODE==2):
        temp_df_x.append(df['A131_W'].iloc[i])
    else:
        temp_df_x.append(np.empty(np.array(df['A131_W'].iloc[i]).shape))
        temp_df_x[i][:] = np.NaN
temp_df_x = np.array(temp_df_x)

outX_b = getData(temp_df_x)

temp_df_y = []
for i in range(len(df)):
    if(df.iloc[i].WORKMODE==2):
        temp_df_y.append(df['A132_W'].iloc[i])
    else:
        temp_df_y.append(np.empty(np.array(df['A132_W'].iloc[i]).shape))
        temp_df_y[i][:] = np.NaN
temp_df_y = np.array(temp_df_y)

outY_b = getData(temp_df_y)


temp_df_z = []
for i in range(len(df)):
    if(df.iloc[i].WORKMODE==2):
        temp_df_z.append(df['A133_W'].iloc[i])
    else:
        temp_df_z.append(np.empty(np.array(df['A133_W'].iloc[i]).shape))
        temp_df_z[i][:] = np.NaN
temp_df_z = np.array(temp_df_z)

outZ_b = getData(temp_df_z)



freq=2000 #choose frequency 100,500,1700,2000 kHz
row=frequency(freq)
outX_b=Amplitude2(outX_b)
outY_b=Amplitude2(outY_b)
outZ_b=Amplitude2(outZ_b)

df_burst=pd.DataFrame(list(zip(outX_b[row,:],outY_b[row,:],outZ_b[row,:],GEO_LAT,GEO_LON)),columns =[f'EFDX_Amplitude_burst zone_{freq}Hz_from waveform_[0-{len(S_burst.W

df[f'EFDX_Amplitude {freq}Hz_[0-{powerX.shape[0]}]_from power spectrum whole orbit']=powerX.T[row,:].tolist()
df[f'EFDY_Amplitude {freq}Hz_[0-{powerY.shape[0]}]_from power spectrum whole orbit']=powerY.T[row,:].tolist()
df[f'EFDZ_Amplitude {freq}Hz_[0-{powerZ.shape[0]}]_from power spectrum whole orbit']=powerZ.T[row,:].tolist()

max_global_meanX = 580
min_global_meanX = 220
max_global_meanY = 580
min_global_meanY = 220
max_global_meanZ = 580
min_global_meanZ = 220
max_global_meanXb = 580
min_global_meanXb = 220
max_global_meanYb = 580
min_global_meanYb = 220
max_global_meanZb = 580
min_global_meanZb = 220

MakeGraph(f'EFDX_Amplitude {freq}Hz_[0-{powerX.shape[0]}]_from power spectrum whole orbit', df,
          max_global_meanX, min_global_meanX)
MakeGraph(f'EFDY_Amplitude {freq}Hz_[0-{powerY.shape[0]}]_from power spectrum whole orbit', df,
          max_global_meanY, min_global_meanY)
MakeGraph(f'EFDZ_Amplitude {freq}Hz_[0-{powerZ.shape[0]}]_from power spectrum whole orbit', df,
          max_global_meanZ, min_global_meanZ)

MakeGraph(f'EFDX_Amplitude_burst zone_{freq}Hz_from waveform_[0-{len(S_burst.WORKMODE)}]', df_burst,
          max_global_meanXb, min_global_meanXb)
MakeGraph(f'EFDY_Amplitude_burst zone_{freq}Hz_from waveform_[0-{len(S_burst.WORKMODE)}]', df_burst,
          max_global_meanYb, min_global_meanYb)
MakeGraph(f'EFDZ_Amplitude_burst zone_{freq}Hz_from waveform_[0-{len(S_burst.WORKMODE)}]', df_burst,
          max_global_meanZb, min_global_meanZb)
```