

# SigMT - User Manual

v. 1.1

K.S. Ajithabh

January 6, 2023

## Introduction

SigMT is a python package designed for the processing of the raw magnetotelluric (MT) data to obtain the MT impedance and tipper estimates. It works in an automated way, so that manual time series inspection and editing are not required. Mahalanobis based data selection tool is implemented in the package to avoid the manual editing of time series. The final impedance estimation is done using the robust estimation method. Different data selection tools such as coherency threshold, polarization direction are included in this package. This document is a guide to use the SigMT package. At present, only ‘.ats’ file formats from ADU07 (Metronix Geophysics) is supported.

# Contents

1	Supported file formats	3
2	Getting started	3
3	Overview of the package	4
4	Setting of parameters	7
5	An example	8
5.1	Running the program . . . . .	8
5.2	Creating EDI file . . . . .	10
5.3	Data selection tools . . . . .	15
5.4	Remote reference . . . . .	17

# 1 Supported file formats

- \*.ats file format from ADU-07 by Metronix Geophysics

## 2 Getting started

The program is written as a project in Spyder IDE with the support of ‘Anaconda’ Python Distribution. I suggest installation of the Anaconda first to start with this package. You can download Anaconda for Windows/Linux/Mac from Anaconda website. Please install Anaconda3-2021.11 version from the Anaconda archives using the link <https://repo.anaconda.com/archive/>. Because newer version is showing some errors. Anaconda provides all necessary python modules for the scientific computations.

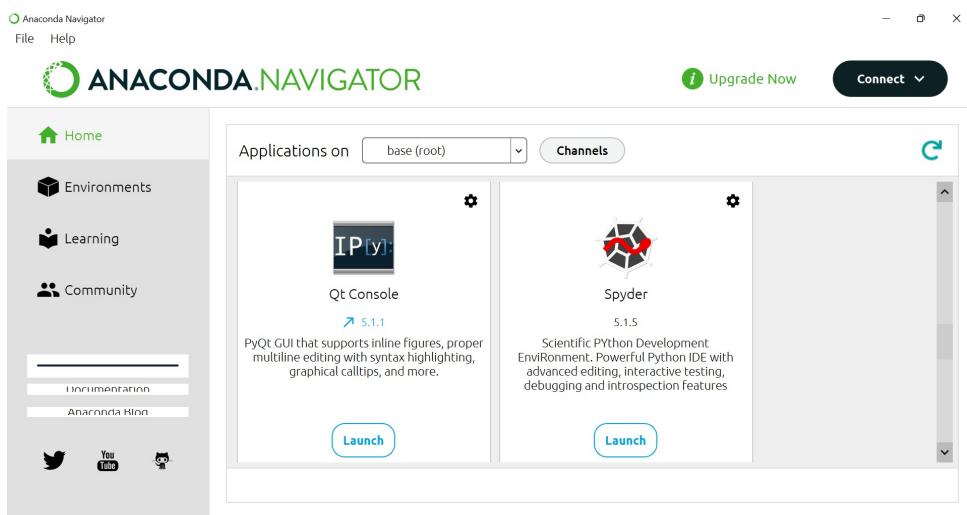


Figure 1: Anaconda Navigator

After installation of Anaconda, you can check in Anaconda Navigator (Figure 1) whether Spyder is installed or not. If not installed by default, please install Spyder using install button in Anaconda Navigator.

Once Anaconda and Spyder is ready, you can download the SigMT package as zip from GitHub (Figure 2). Extract the compressed folder to your local drive. Then open ‘Spyder IDE’, select Projects and click ‘Open Project’ (Figure 3).

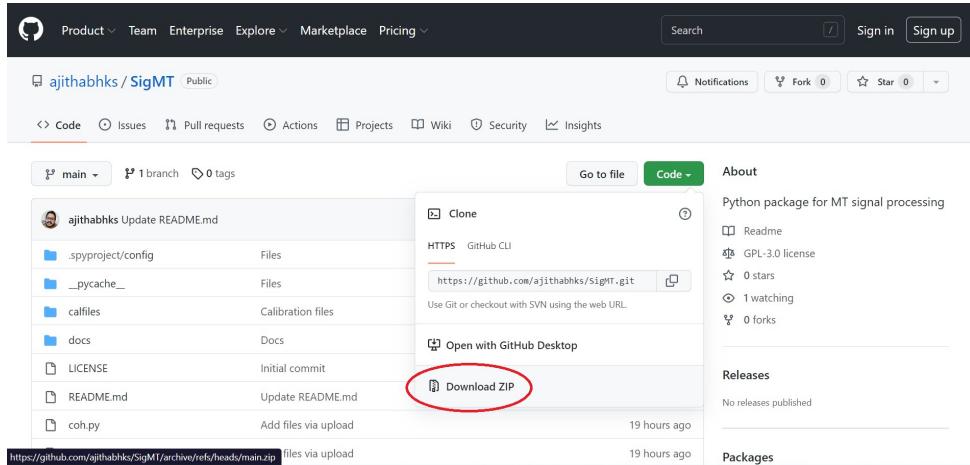


Figure 2: Download the package

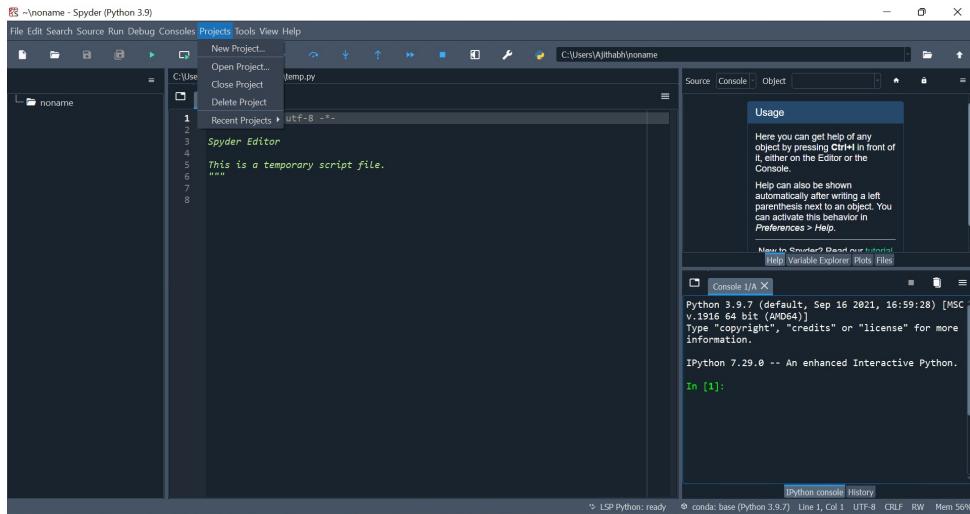


Figure 3: Open Project in Spyder

Navigate to extracted SigMT project folder and select it. Now you are in SigMT package and all set to start processing your MT data!!

### 3 Overview of the package

The file ‘main\_script.py’ is the file you need to run the package for single site processing. But, if you need to carry out Remote reference, please use ‘main\_script\_RR.py’ file.

The files such as mtproc.py, mtprocRR.py, var.py, mahaDist.py, data\_sel.py, tipper.py are written to perform different tasks in package and need not to be edited.

I suggest editing of main\_script.py, main\_script\_RR.py and config.py as per your needs. The more description of these files are given below.

First of all, create a project folder and copy all your MT sites in it (as in Figure 4).

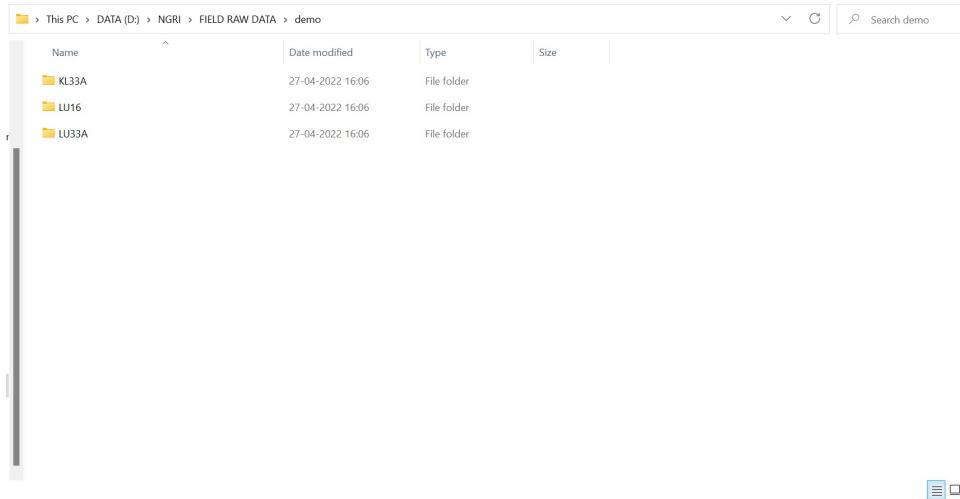


Figure 4: Project folder

The measurements should be in the site folder as in Figure 5. There should not be any other extra folders inside the site folder.

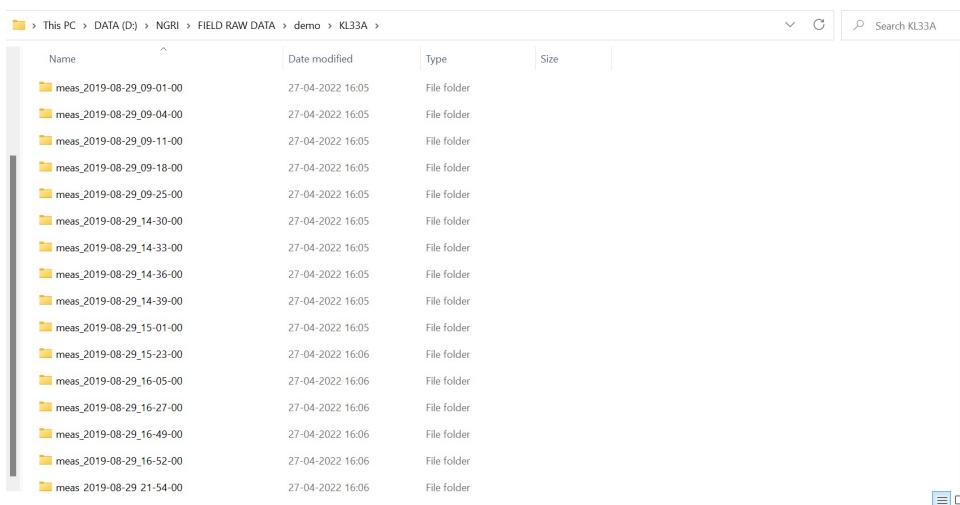


Figure 5: Site folder

There are three places, code can be modified in ‘main\_script.py’ and ‘main\_script\_RR.py’ files to provide site folder path, decimation and data selection constraints.

First edit is required to provide the sites folder path. Copy the location of folder where sites are kept and copy to the variable ‘project\_path’ as in Figure 6.

```

D:\Pyth\demo - Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Pyth\demo\main_script.py
main_script.py
1  # -*- coding: utf-8 -*-
2  """
3      Created on Mon May  4 16:49:35 2020
4
5      @author: AJITHABH
6  """
7  import mtproc, coh, coherence, tipper, mahaDist
8  from scipy import signal
9  from matplotlib import pyplot as plt
10 import numpy as np
11 import os
12 import time
13 import math
14 #
15 #
16 # provide project path where sites are kept
17 project_path = 'D:/NGRI/FIELD RAW DATA/demo/' (highlighted)
18 #
19 s.chdir(project_path)
20 sites = [d for d in os.listdir('..') if os.path.isdir(d)]
21 # all site names are stored in variable sites
22 print('Sites in the project are: ')
23 print(sites)
24 selectedsite = input("Enter the site: ")
25 siteindex = sites.index(selectedsite)
26 measid = mtproc.measid(siteindex)
27 del siteindex
28 os.chdir(project_path[selectedsite])
29 all_meas = [d for d in os.listdir('..') if os.path.isdir(d)]
30 for i in range(len(all_meas)):
31     print(all_meas[i])

```

Figure 6: Give project path (sites folder)

Second edit is required in case you require decimation. The highlighted portions in Figure 7 is written to perform decimation. Keep ‘dflag’ 1 if you need to used decimation, else always keep 0. Suppose you need to decimate the data sampled at 32 Hz. If you use [8,4], then the data will be first decimated to  $32/8 = 4$  Hz, then  $4/4 = 1$  Hz. So, the output will be 1 Hz. Direct decimation using [32] is not recommended. If you used [8,8,4], the data will be decimated to 0.125 Hz. Similarly, you can decimate to any sampling frequencies.

```

D:\Pyth\SigMT-Distribution - Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Pyth\SigMT-Distribution\main_script.py
main_script.py
43 #
44 # ===== Selection of site and setting a path =====
45 sites, selectedsite, measid, all_meas, select_meas, proc_path = mtproc.makeprocpath(project_pat
46 #
47 # ===== Site is selected and path is created =====
48 #
49 # ===== Time series reading starts =====
50 procinfo = {}
51 ts, procinfo['fs'], procinfo['sensor_no'], timeline, procinfo['ChoppStat'], loc = mtproc.ts(pro
52 #===== Decimation section =====
53 # Keep dflag = 0 if decimation is not required
54 dflag = 0
55 if dflag == 1:
56     decimate = [8,8,4]
57     for d in decimate:
58         ts['tsEx'] = signal.decimate(ts.get('tsEx'), d, n=None, ftype='fir')
59         ts['tsEy'] = signal.decimate(ts.get('tsEy'), d, n=None, ftype='fir')
60         ts['tsHz'] = signal.decimate(ts.get('tsHz'), d, n=None, ftype='fir')
61         ts['tsHy'] = signal.decimate(ts.get('tsHy'), d, n=None, ftype='fir')
62         ts['tsNx'] = signal.decimate(ts.get('tsNx'), d, n=None, ftype='fir')
63         ts['tsNy'] = signal.decimate(ts.get('tsNy'), d, n=None, ftype='fir')
64         procinfo['fs'] = procinfo.get('fs')/d
65 #===== Decimation section end =====
66 #
67 # Some calculations and printing some information
68 # No need to edit
69 procinfo['nofs'] = len(ts['tsEx'])
70 procinfo['notch'] = 0 # Notch flag 1 - On, 0 - Off
71 print('-----')
72 print('Site: ' + selectedsite)
73 print('Measurement directory: ' + all_meas[select_meas])
74 print('fs: ' + str(procinfo.get('fs')) + ' Hz')
75 print('no. of measurement numbers: ' + str(procinfo['nofs']))

```

Figure 7: Decimation

Third edit is required to control data selection tools. The highlighted portions in Figure 8 is written to perform data selection. Coherency threshold and polarization direction based selections are included in the package. More detailed description is given in section 5.3.

```

121 alpha_degX, alpha_degY = data_sel.pdvalues(bandavg)
122 #----- Coherency threshold -----
123 CoThre = [0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7, 0.7]
124 if pdflag == 1: # Give '1' to perform coherency threshold based selection
125     for i in range(np.shape(AlcohEx)[0]):
126         for j in range(np.shape(AlcohEx)[1]):
127             if AlcohEx[i,j] < CoThre[1]:
128                 coMatrixEx[i,j] = 0
129             else:
130                 coMatrixEx[i,j] = 1
131     if pdflag == 0: # Give '0' to perform polarization direction based selection
132         for i in range(np.shape(AlcohEx)[0]):
133             if AlcohEx[i,j] < CoThre[0]:
134                 coMatrixEx[i,j] = 0
135             else:
136                 coMatrixEx[i,j] = 1
137 #----- Polarization direction -----
138 if pdflag == 0: # Give '1' to perform polarization direction based selection
139     for i in range(np.shape(pdmat)[0]):
140         for j in range(np.shape(pdmat)[1]):
141             alpha = alpha_degX # Use either alpha_degX or alpha_degY
142             if i in range(np.shape(pdmat)[0]):
143                 for j in range(np.shape(pdmat)[1]):
144                     if alpha > pdmat[i,j] and alpha[i,j] < pdlim[1]:
145                         pdmat[i,j] = 0
146                     else:
147                         pdmat[i,j] = 1
148     if pdflag == 0: # Give '1' to perform polarization direction based selection
149         for i in range(np.shape(pdmat)[0]):
150             for j in range(np.shape(pdmat)[1]):
151                 if j > timeswindow_limits[0] and j < timeswindow_limits[1]:
152                     pdmat[i,j] = 0
153                 else:
154                     pdmat[i,j] = 1
155

```

Figure 8: Data selection tools section

## 4 Setting of parameters

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Mon Jul 25 11:46:56 2022
4
5 @Author: AJITHABH K. S.
6 Last modified: 25-07-2022
7
8 This is the configuration file to change FFT Length, Parzen Radius,
9 Mahalanobis Distance (MD) threshold for impedance and tipper calculations.
10 """
11
12 def configuration():
13     config = {}
14     config['FFT_Length'] = 0
15     # Give 0 (default value) to automatically select an FFT length according to Borah & Patro,
16     # Otherwise give 256, 512, 1024, 2048, 4096, 8192, 16384
17     config['parzen_radius'] = 0
18     # Give 0 (default value) to select parzen radius according to Borah & Patro, 2015
19     config['MD_threshold_impedance'] = 1.5
20     # Mahalanobis distance threshold value for impedance calculations
21     config['MD_threshold_tipper'] = 1.0
22     # Mahalanobis distance threshold value for tipper calculations
23
24     return config

```

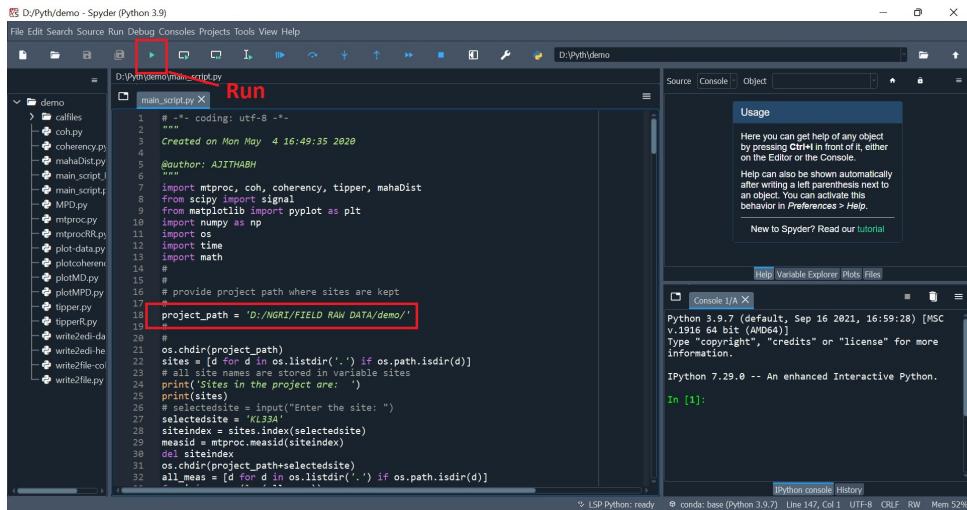
Figure 9: Configuration file

The user can define parameters such as FFT length, parzen radius, mahalanobis distance for impedance calculation, mahalanobis distance for tipper calculation. The configuration file is shown in the Figure 9. Keep FFT\_Length and parzen\_radius as 0 to choose default values. Or else, provide values as per the need.

# 5 An example

## 5.1 Running the program

I am showing an example of processing here. Give the sites folder path in ‘main\_script.py’ file and run the code as in the Figure 10



```
# -*- coding: utf-8 -*-
"""
Created on Mon May 4 16:49:35 2020
@author: AJITHABH
"""

import mproc, coh, coherency, tipper, mahaDist
from scipy import signal
from matplotlib import pyplot as plt
import numpy as np
import os
import time
import math

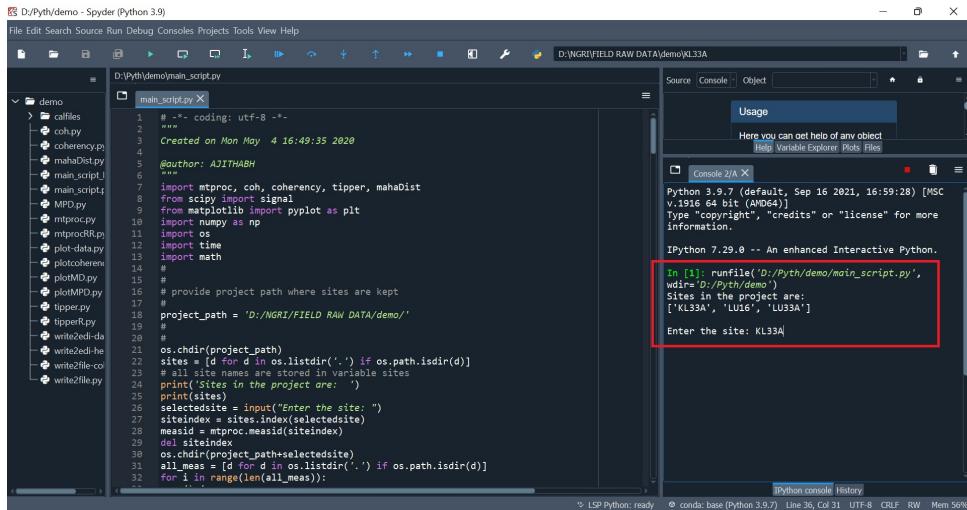
# provide project path where sites are kept
project_path = 'D:/NGRI/FIELD RAW DATA/demo/'

os.chdir(project_path)
sites = [d for d in os.listdir('.') if os.path.isdir(d)]
# all site names are stored in variable sites
print('Sites in the project are: ')
print(sites)

selectedsite = input("Enter the site: ")
selectedsite = 'KL33A'
siteindex = sites.index(selectedsite)
measid = mproc.measid(siteindex)
del siteindex
os.chdir(project_path+selectedsite)
all_meas = [d for d in os.listdir('.') if os.path.isdir(d)]
```

Figure 10: Give path and run the program

Once we run the program, the names of all sites will be displayed in the console (Figure 11). Then enter the site name to be processed and click ‘Enter’.



```
# -*- coding: utf-8 -*-
"""
Created on Mon May 4 16:49:35 2020
@author: AJITHABH
"""

import mproc, coh, coherency, tipper, mahaDist
from scipy import signal
from matplotlib import pyplot as plt
import numpy as np
import os
import time
import math

# provide project path where sites are kept
project_path = 'D:/NGRI/FIELD RAW DATA/demo/'

os.chdir(project_path)
sites = [d for d in os.listdir('.') if os.path.isdir(d)]
# all site names are stored in variable sites
print('Sites in the project are: ')
print(sites)

selectedsite = input("Enter the site: ")
selectedsite = sites.index(selectedsite)
measid = mproc.measid(siteindex)
del siteindex
os.chdir(project_path+selectedsite)
all_meas = [d for d in os.listdir('.') if os.path.isdir(d)]
for i in range(len(all_meas)):
```

Figure 11: Enter the site name from the list

Then all measurements in the selected site will be displayed. Give the measurement number as input. For example, I selected number 16 for the processing in Figure 12.

```

D:\Pyth\demo - Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Pyth\demo\main_script.py
Source Console Object
Usage
Here you can get help of any object
Help Variable Explorer Plots Files
Console 2/A X
Sites in the project are:
['KL33A', 'LU16', 'LU33A']

Enter the site: KL33A
[[0, 'meas_2019-08-29_09-01-00'],
 [1, 'meas_2019-08-29_09-04-00'],
 [2, 'meas_2019-08-29_09-11-00'],
 [3, 'meas_2019-08-29_09-18-00'],
 [4, 'meas_2019-08-29_09-25-00'],
 [5, 'meas_2019-08-29_14-06-00'],
 [6, 'meas_2019-08-29_14-13-00'],
 [7, 'meas_2019-08-29_14-36-00'],
 [8, 'meas_2019-08-29_14-59-00'],
 [9, 'meas_2019-08-29_15-01-00'],
 [10, 'meas_2019-08-29_15-23-00'],
 [11, 'meas_2019-08-29_16-05-00'],
 [12, 'meas_2019-08-29_16-28-00'],
 [13, 'meas_2019-08-29_16-49-00'],
 [14, 'meas_2019-08-29_16-52-00'],
 [15, 'meas_2019-08-29_21-54-00'],
 [16, 'meas_2019-08-29_21-57-00'],
 [17, 'meas_2019-08-29_22-19-00']]

Select measurement: 16

```

Figure 12: Enter the measurement number

Then some details about the measurement such as sampling frequency, coil numbers, length of time series, window length, and number of stacks will be displayed (Figure 13). Then the processing will be started.

```

D:\Pyth\demo - Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Pyth\demo\main_script.py
Source Console Object
Usage
Here you can get help of any object
Help Variable Explorer Plots Files
Console 2/A X
[14, 'meas_2019-08-29_22-19-00']

Select measurement: 16
Basic details
NT site: KL33A
Measurement directory: meas_2019-08-29_21-57-00
fs=4096.0 Hz
Sensor numbers:
{'Hx': [300], 'Hy': [301], 'Hz': [302]}
Length of time series = 4915200
-----
Unused variables deleted.
-----
Window Length selected: 16384
Time series overlap: 50%
No. of stacks: 599
-----
Band averaging over target frequencies:
35|| , 17/599 [08:09:05;13, 1.86it/s]

```

Figure 13: Processing..

After completing the processing, figure showing apparent resistivity and phase curves, coherency values and tipper data will be plotted and can be seen in ‘Plots’ section (Figure 14).

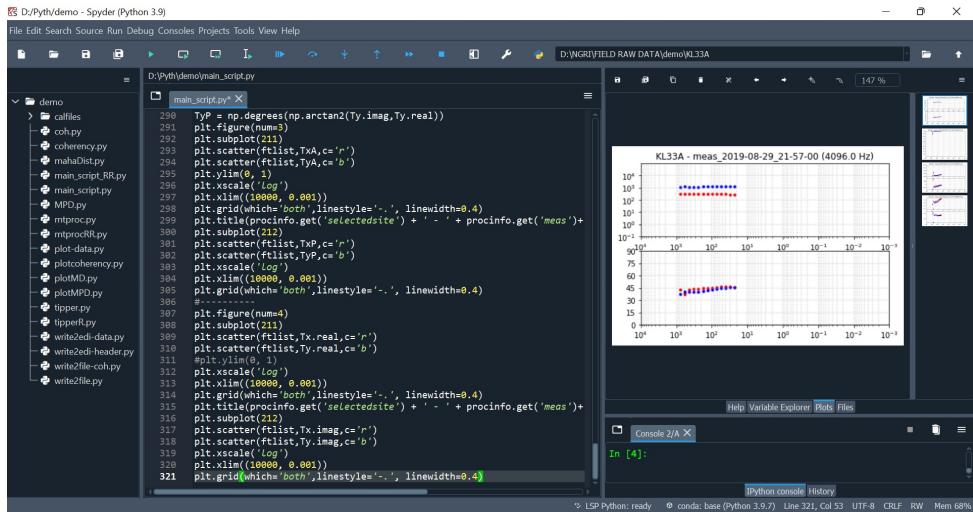


Figure 14: Output figures

## 5.2 Creating EDI file

The data is processed band wise. After completing processing for all measurements, the data should be joined to create the EDI files. One site may have many measurements. So, we have to save data in text files.

Create a folder anywhere. For example, a folder named ‘Outputs’ (as in Figure 15). Inside that folder, make a folder with site name. Create a folder named ‘bands’ inside site folder (Figure 15).

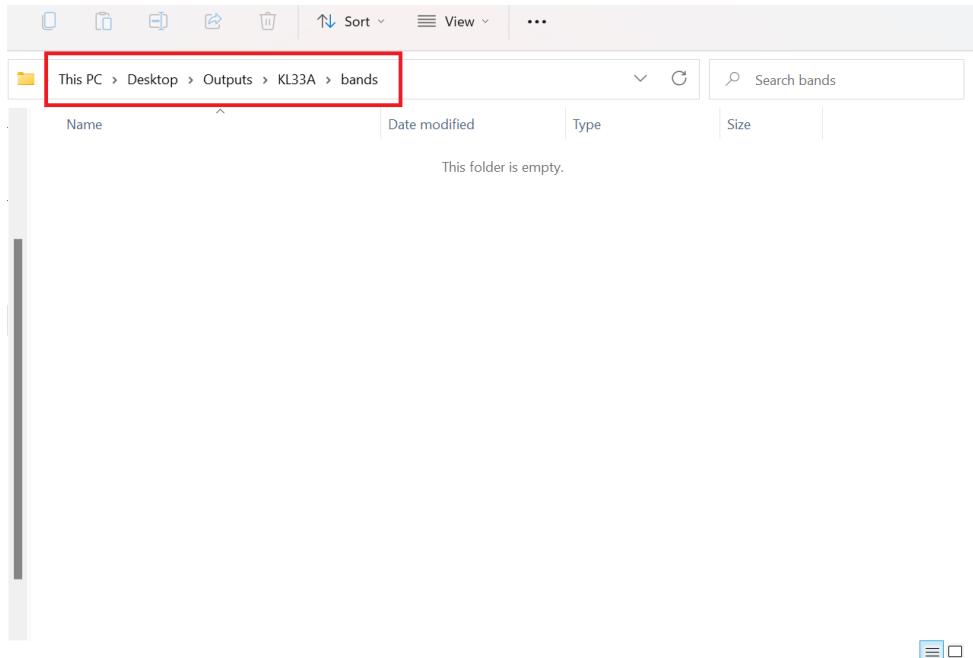


Figure 15: Create a folder to save processed data

Now come to Spyder and open ‘write2file.py’ script. Give the path to bands folder in

the variable ‘f’. Give sampling frequency as file name. For example ‘4096Hz.txt’. Select all script lines using ‘Ctrl+A’ and right click. Select ‘Run section or current line’ (Figure 16).

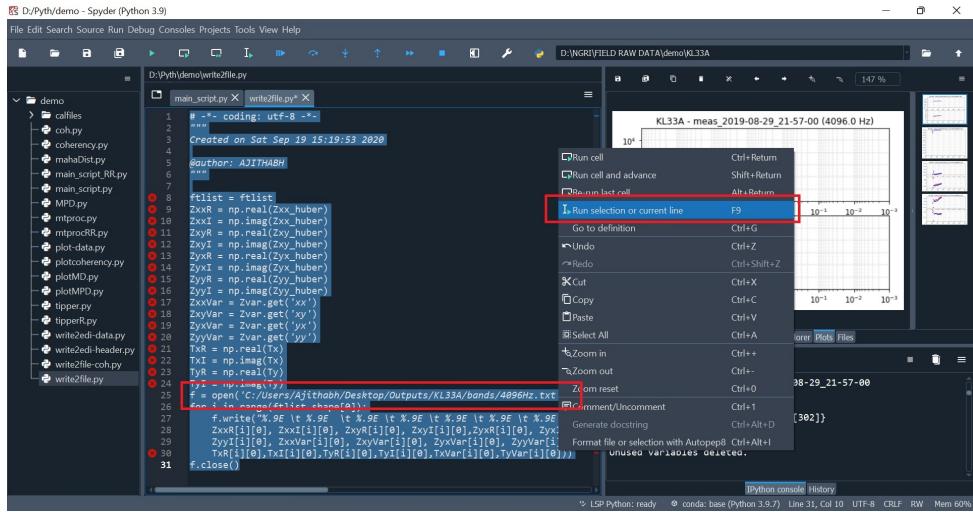


Figure 16: Save data in file

Now save header information to a file in site output folder. It is required for EDI file creation. Open ‘write2edi-header.py’ file and give path to sites folder (output) in variable ‘f’. Select all lines, right click and select ‘Run section or current line’ (Figure 17)

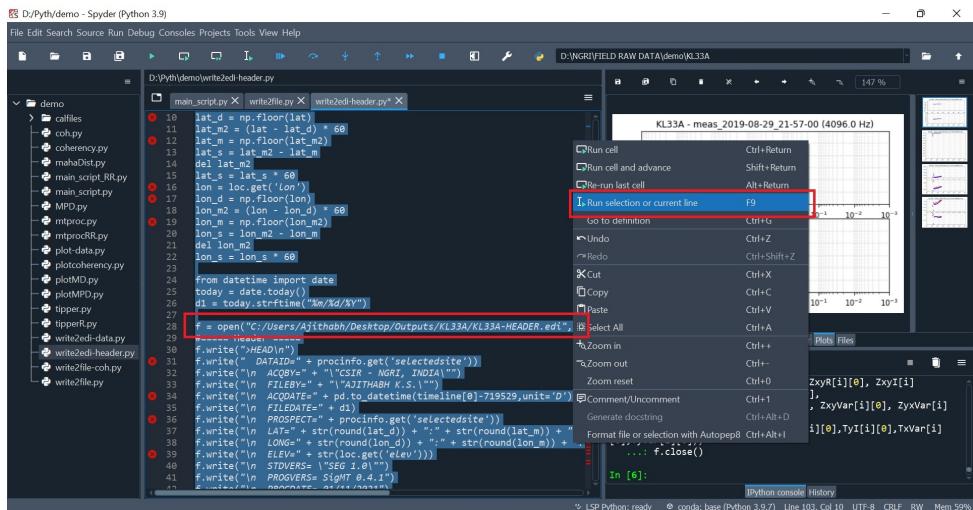


Figure 17: Save header information

After executing processing for a measurement, always close ‘Console’ (Figure 18).

The screenshot shows the Spyder Python IDE interface. On the left, there's a file tree for a 'demo' folder containing various Python scripts like 'coh.py', 'coherence.py', 'mahaDist.py', etc. Two scripts are open in editors: 'main\_script.py' and 'write2file.py'. The code in 'main\_script.py' includes imports for numpy, pandas, and procinfo, along with logic to calculate latitudes and longitudes from a location object. It also includes code to open a file at a specific path and write data to it. The code in 'write2file.py' is a copy of the code from 'main\_script.py'.

On the right, there's a plot window titled 'KL33A - meas\_2019-08-29\_21-57-00 (4096.0 Hz)'. The plot shows a scatter of data points on a logarithmic scale for both axes, ranging from 10<sup>-4</sup> to 10<sup>4</sup>. Below the plot, a 'Console 2/A' window shows the command to write the file 'all.txt'.

Figure 18: Close console after processing a measurement

Similarly, I processed for 1024 Hz measurement and save data in bands folder of KL33A. Now the measurement data can be seen in the folder (Figure 19). Create a text file named 'all.txt'.

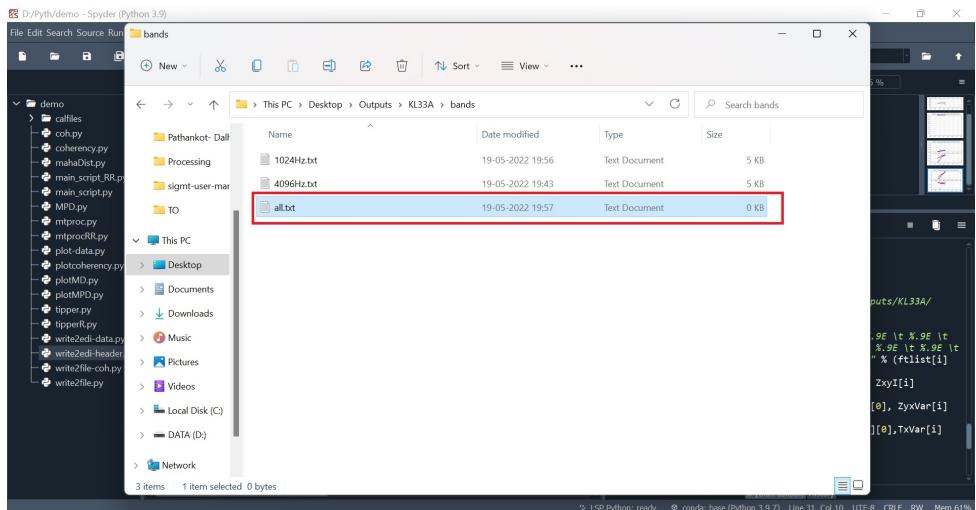


Figure 19: Create a file all.txt

Copy the data from 4096Hz.txt file and 1024Hz.txt file to all.txt as in Figure 20. Always keep higher frequency data in the top.

The screenshot shows a Notepad window titled 'all.txt - Notepad'. The content of the file is a large block of numerical data, mostly in scientific notation. Several lines are highlighted in red, specifically:

- Line 1:** 4096 F1
- Line 2:** 1024 M1
- Line 3:** 1024 M1

The rest of the data consists of approximately 1000 lines of similar numerical values.

Figure 20: Copy data to all.txt file

Now, again go to Spyder and open ‘write2edi-data.py’ file. Give file path to ‘all.txt’ and path of sites folder in variables ‘edifilename’ and ‘f’ (Figure 21).

The screenshot shows the Spyder IDE interface with the file 'write2edi-data.py' open. The code is as follows:

```

1 # -*- coding: utf-8 -*-
2 """
3 Created on Wed Jan 13 15:47:38 2021
4
5 @author: AJITHABH
6
7
8 import pandas as pd
9 import numpy as np
10
11 edifilename = 'C:/Users/Ajithabh/Desktop/Outputs/KL33A/bands/LL1.EDI'
12 f = open('C:/Users/Ajithabh/Desktop/Outputs/KL33A/KL33A-04TA.EDI', 'w')
13
14 data = pd.read_csv(edifilename, sep='\t', lineterminator='\n')
15 data = np.asarray(data)
16 freq = data[:,0]
17 zxxr = data[:,1]
18 zxyr = data[:,2]
19 zxxy = data[:,3]
20 zxyl = data[:,4]
21 zyxr = data[:,5]
22 zyxl = data[:,6]
23 zyyr = data[:,7]
24 zyyl = data[:,8]
25 zxvar = data[:,9]
26 zxxyar = data[:,10]
27 zyxyar = data[:,11]
28 zyyvar = data[:,12]
29 Tyr = data[:,13]
30 Txy = data[:,14]
31 Tyr = data[:,15]
32 TyI = data[:,16]

```

A context menu is open over the line 'f = open('C:/Users/Ajithabh/Desktop/Outputs/KL33A/KL33A-04TA.EDI', 'w')'. The 'Run selection or current line' option is highlighted.

Figure 21: Write data as in EDI

Now, two files are existing in sites folder (Figure 22). Copy data in the file KL33A- DATA.edi to the end of header information in KL33A-HEADER.edi file (Figure 23). Please correct ‘NFREQ’ variable with actual number of frequencies in the data (Figure 23). Now, KL33A-HEADER.edi is a standard EDI file ready to use.

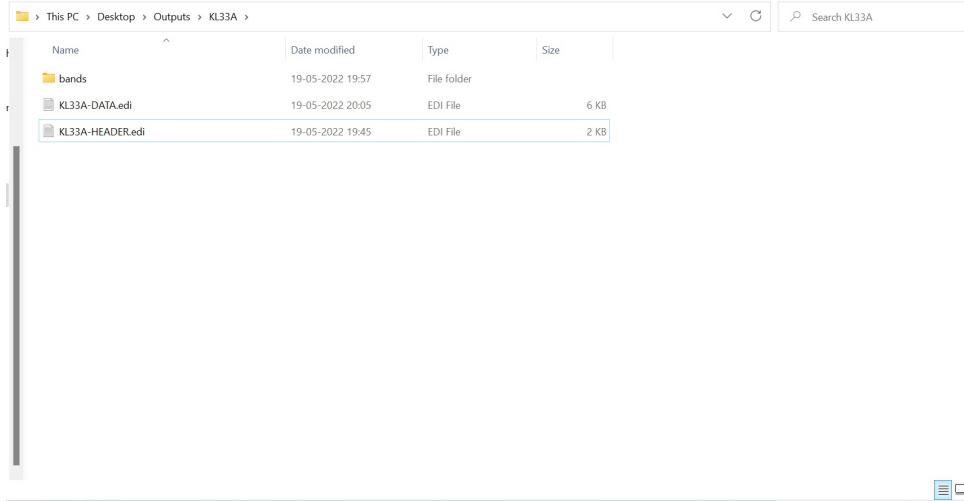


Figure 22: Files ready to create EDI

```
*KL33A-HEADER.edi - Notepad
File Edit View
>HMEAS ID=4.001 CHTYPE=HY X=0.00000000E+00
Y=0.00000000E+00 Z=0.00000000E+00
ACQCHAN="ADU07/MFS06 /0/" GAIN=1 MEASDATE=08/29/2019
AZH=9.00000000E+01 DIP=0.00000000E+00
SENSOR=MFS06 /
>HMEAS ID=5.001 CHTYPE=HZ X=0.00000000E+00
Y=0.00000000E+00 Z=0.00000000E+00
ACQCHAN="ADU07/MFS06 /0/" GAIN=1 MEASDATE=08/29/2019
AZH=0.00000000E+00 DIP=0.00000000E+00
SENSOR=MFS06 /
>HTSCT
>SECTID=KL33A
NFREQ=13
HX=3.001
HY=4.001
HZ=5.001
EX=1.001
EY=2.001
>FREQ //17
5.7103854095E+02 4.258777131E+02 3.176174875E+02 2.368775478E+02 1.766621010E+02
5.317537193E+02 9.826127078E+01 3.28275869E+01 5.465392565E+01 4.076062685E+01
3.039907346E+01 2.267147831E+01 1.698827615E+01 1.261810854E+01 9.404556446E+00
7.013871582E+00 5.230910660E+00
>ZXXR //17
-7.023137803E+01 1.020347167E+01 1.315706979E+01 1.655562791E+01 1.870547240E+01
2.023157685E+01 1.7840000015E+01 2.158549529E+01 2.051765238E+01 1.725275253E+01
1.361852426E+01 5.388805539E+00 -4.158872013E-01 -2.430117559E+00 -2.661894916E+00
-1.072436996E-02 5.565941568E+00
Ln 58, Col 11
100% Windows (CRLF) UTF-8
```

Figure 23: Edit NFREQ and save EDI

### 5.3 Data selection tools

```

D:\Pyth\SigMT-Distribution - Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Pyth\SigMT-Distribution\main_script.py
main_script.py
121 alpha_degE, alpha_degT = data_sel.pdvalues(bandavg)
122
123 ##### Coherency threshold
124 if ctflag == 1:
125     CohTre = [0.7,0.7,0.7,0.7,0.7,0.7,0.7,0.7,0.9,0.9,0.9]
126     for i in range(mp.shape(AllcohExy)[1]):
127         for j in range(mp.shape(AllcohExy)[1]):
128             if AllcohExy[i,j] < CohTre[i]:
129                 cohMatrixEx[i,j] = 0
130             else:
131                 cohMatrixEx[i,j] = 1
132
133             cohMatrixEy[i,j] = 1
134             if AllcohEy[i,j] < CohTre[i]:
135                 cohMatrixEy[i,j] = 0
136             else:
137                 cohMatrixEy[i,j] = 1
138
139 ##### Polarization direction #####
140 pdflag = 0 # Give '1' to perform polarization direction based selection
141 if pdflag == 1:
142     pdlim = [40,60]
143     alpha = alpha_degE + (no_sides*alpha_degT or alpha_degT
144     for i in range(mp.shape(alpha)[1]):
145         for j in range(mp.shape(pdmat)[1]):
146             if alpha[i,j] > pdlim[0] and alpha[i,j] < pdlim[1]:
147                 pdmat[i,j] = 0
148             else:
149                 pdmat[i,j] = 1
150
151 pdflag = 0 # Give '1' to perform polarization direction based selection
152 if pdflag == 1:
153     timewindow_limits = [0,6]
154     for i in range(mp.shape(pdmat)[0]):
155         for j in range(mp.shape(pdmat)[1]):
156             if j > timewindow_limits[0] and j < timewindow_limits[1]:
157                 pdmat[i,j] = 0
158             else:
159                 pdmat[i,j] = 1

```

Figure 24: Data selection tools section

In Figure 24, we can see the data selection tools part of package. If `ctflag` is 1, the coherency threshold will be activated. The coherency threshold value can be provide in ‘`CohTre`’ variable for each target frequency. You can see the coherency values for each target frequency using the ‘`plot-coherency.py`’ script. Give `Z_all = bandavg.get('Zxy_single')` to see coherency of xy component and `Z_all = bandavg.get('Zyx_single')` for yx component. Once you run the script, coherency plots will be displayed for each target frequency as in Figure 26. By analysing the values in the figure, you can set coherency threshold values.

```

D:\Pyth\demo - Spyder (Python 3.9)
File Edit Search Source Run Debug Consoles Projects Tools View Help
D:\Pyth\demo\plotcoherency.py
main_script.py X plotcoherency.py
1: ing: utf-8 -*- 0
2: n Thu Aug 20 15:52:42 2020
3: AJITHABH
4:
5: tplotlib
6: ib.use('TKAgg')
7: lolib importplotlib as pl
8: red: ((0.0, 0.0, 0.0),
9:        (0.1, 0.0, 0.5),
10:       (0.2, 0.0, 0.0),
11:       (0.4, 0.2, 0.2),
12:       (0.6, 0.6, 0.0),
13:       (0.8, 1.0, 1.0))
14: green: ((0.0, 0.0, 0.0),
15:        (0.1, 0.0, 0.0),
16:        (0.2, 0.0, 0.0),
17:        (0.4, 1.0, 1.0))
18: blue: ((0.0, 0.0, 0.0),
19:        (0.1, 0.5, 0.5),
20:        (0.2, 1.0, 1.0),
21:        (0.6, 1.0, 1.0),
22:        (0.8, 1.0, 1.0),
23:        (1.0, 0.0, 0.0))
24: matplotlib.colors.Linear
25:
26:
27:
28:
29:
30:
31:
32:

```

Context menu for the 'green' color definition:

- Run cell
- Run cell and advance
- Re-run last cell
- Run selection or current line (highlighted)
- Go to definition
- Undo
- Redo
- Cut
- Copy
- Paste
- Select All
- Zoom in
- Zoom out
- Zoom reset
- Comment/Uncomment
- Generate docstring
- Format file or selection with Autoprefixer

Figure 25: Run codes

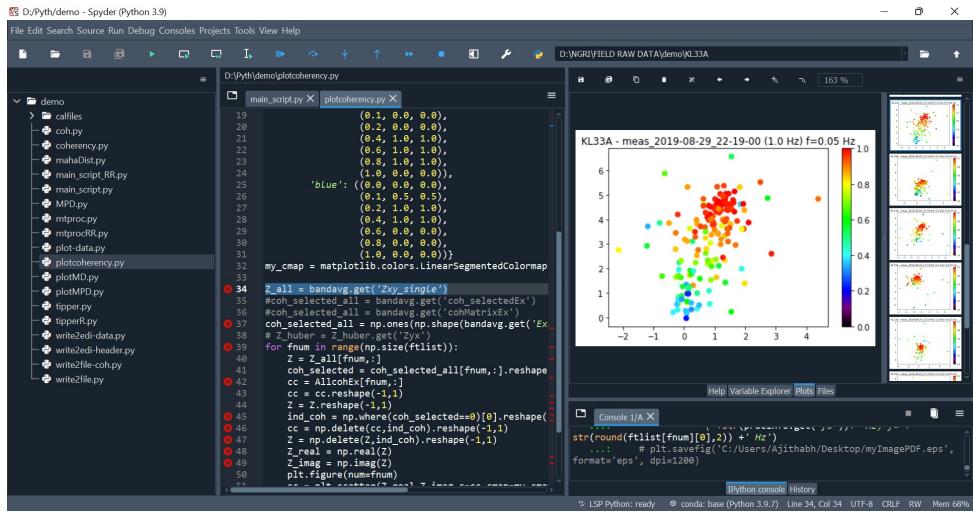


Figure 26: Coherency values

As similar in the case of coherency threshold, run scripts in ‘plot-pd.py’ to plot polarization directions for each stacks for all target frequencies (Figure 27). The top panel in the plot shows magnetic polarization directions and bottom panel shows electric polarization directions. Analysing the plots, we can identify the polarized segments. Polarization directions can be selected in two ways. We can select a range of polarization direction to be discarded in the case 1. All stacks with polarization direction values [-10,10] will be discarded in this case. In case you need to use magnetic polarization direction, use ‘alpha\_degH’ and for electric polarization direction, use ‘alpha\_degE’ (as in Figure 28).

Next is to select stacks, give a range of stacks to be discarded in this case. In the example (Figure 28), the stacks between 300 and 403 will be discarded from processing.

‘pdflag’ should be 1 to perform the tasks.

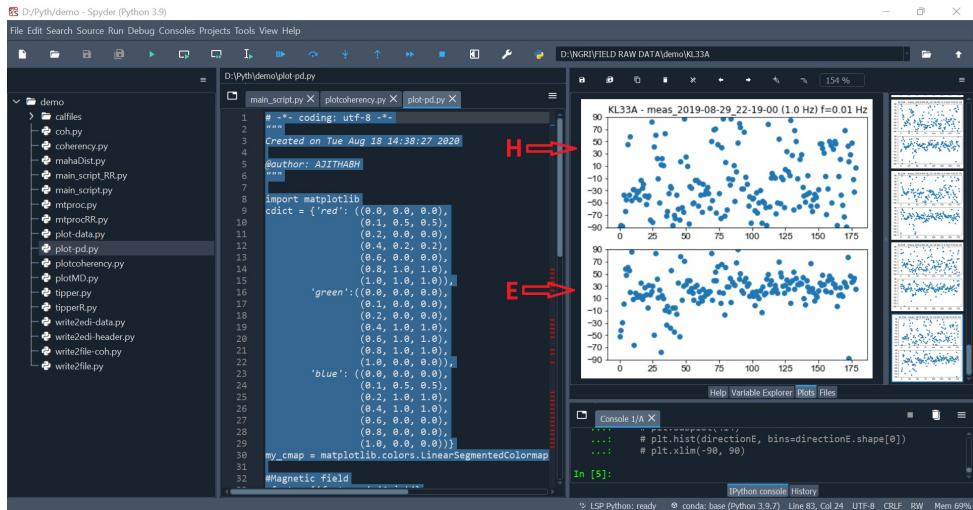


Figure 27: Polarization directions

The screenshot shows the Spyder IDE interface with the file 'main\_script.py' open. The code handles polarization direction selection based on various flags and thresholds. A specific section of the code is highlighted in red:

```

130     cohMatrixEx[i,j] = 0
131     else:
132         cohMatrixEx[i,j] = 1
133     if AllcohEx[i,j] < CohThre[i]:
134         cohMatrixEx[i,j] = 0
135     else:
136         cohMatrixEx[i,j] = 1
137     # ====== Polarization direction ======
138     pdflag = 0 # Give '1' to perform polarization direction based selection
139     if pdflag == 1:
140         pdlim = [50,60]
141         alpha = alpha_degE # Use either alpha_degE or alpha_degH
142         pdmat = np.zeros(np.shape(pdmat)[0]):
143             for j in range(np.shape(pdmat)[1]):
144                 if alpha[i,j] > pdlim[0] and alpha[i,j] < pdlim[1]:
145                     pdmat[i,j] = 0
146                 else:
147                     pdmat[i,j] = 1
148             # ====== Polarization direction ======
149     pdflag = 0 # Give '1' to perform polarization direction based selection
150     if pdflag == 1:
151         timewindow_limits = [0,5]
152         for i in range(np.shape(pdmat)[0]):
153             for j in range(np.shape(pdmat)[1]):
154                 if j > timewindow_limits[0] and j < timewindow_limits[1]:
155                     pdmat[i,j] = 0
156                 else:
157                     pdmat[i,j] = 1
158             #
159             bandavg[cohMatrixEx] = cohMatrixEx
160             bandavg[cohMatrixEx] = cohMatrixEx
161             bandavg[cohMatrixEx] = cohMatrixEx

```

Figure 28: Magnetic or Electric Polarization direction

Once data selection constraints are set, just run a part of code as highlighted (to end of the script) in figure 29.

The screenshot shows the Spyder IDE interface with the file 'main\_script.py' open. The code includes sections for band averaging and data selection tools. A specific section of the code is highlighted in red:

```

107     ##### Start band averaging =====
108     # Band average to edit
109     # Band average after calibration and averaging using parzen window
110     fflist,bandavg = mproc.bandavg(ts,procinfo.config)
111     #
112     ##### Band averaging finished =====
113     # Data selection tools section, Coherency threshold & Polarization direction
114     cohMatrixE = np.ones(np.shape(bandavg.get('ExExc')),dtype=float)
115     cohMatrixE = np.ones(np.shape(bandavg.get('ExExc')),dtype=float)
116     pdmat = np.ones(np.shape(bandavg.get('ExExc')),dtype=float)
117     # Calculation of coherency values for all time windows
118     AllcohEx = data_sel.coherency(bandavg)
119     AllcohEx = sel.coherency(bandavg)
120     # Calculation of polarization directions for all time windows
121     alpha_degE,alpha_degH = data_sel.pdvalues(bandavg)
122     #
123     ##### Coherency threshold =====
124     ctflag = 0 # Give '1' to perform coherency threshold based selection
125     if ctflag == 1:
126         CohThre = [0.7,0.7,0.7,0.7,0.7,0.7,0.7,0.7,0.7,0.9,0.9,0.9]
127         for i in range(np.shape(AllcohEx)[0]):
128             for j in range(np.shape(AllcohEx)[1]):
129                 if AllcohEx[i,j] < CohThre[i]:
130                     cohMatrixEx[i,j] = 0
131                 else:
132                     cohMatrixEx[i,j] = 1

```

Figure 29: Run this part of script

## 5.4 Remote reference

The remote reference can be done similar as above example. But run 'main\_script\_RR.py' file. First you have to enter the site you need to process and enter the measurement number. Then enter the remote site name and measurement number.