pyCSAMT User Guide

April 19th, 2021

1. Introduction

pyCSAMT is an open source toolkit for controlled source audio-frequency magnetotelluric (CSAMT) standard data processing, visualization and interpretation enhancement written in Python language. Indeed, CSAMT has been recognized as a well-suited geophysical method in a few kilometers' exploration because of its better vertical resolution, wide range of exploration depth. Despite its application in diverse exploration problems such as detecting deep geological structures and groundwater exploration, there is no unified software as open source software for CSAMT data processing and interpretation within the academic community. For that reason, the software is designed to allows handling, processing and imaging of CSAMT far field data sets and also to improve the geophysical interpretation. For more details, we recommend you to follow our wiki pages:
https://github.com/WEgeophysics/pyCSAMT/wiki

2. Important tricks and tips before getting started

This workbook is a set of implementing codes then before starting, we need to show three specials and useful details:

1. pyCSAMT works with three(03) types of data format:
    a. *.*avg* format from Zonge International Engineering company, see our wiki page for more details about zonge *.avg format.
    b. *.*edi* or (Electrical Data Interchange (EDI)) format from Society of Exploration Geophysics (SEG), most recommended in geophysics community
    c. . *. *j* or *.*dat* format of Alan G. Jones (A.G. Jones, October 1994) commonly used in Magnetotelluric domain. See mtnet.info/docs/jformat.txt for more details

    Note: Although the software can read three files, it's strongly recommended to convert both files (*.*avg* and *.*j* files) to SEG *.*edi* files to take advantage of full functionalities of the software.

2. User needs to make sure that pyCSAMT is already installed on his computer and is working perfectly. For installation guide please copy paste the link: https://github.com/WEgeophysics/pyCSAMT/wiki/pyCSAMT-installation-guide-for-Windows--and-Linux for further details.

3. For more details about specific input parameters and other functionalities, we recommend to looking at the software documentation, which can be found at: https://pycsamt.readthedocs.io/en/latest/

Throughout this workbook, the savepath should be None which refer to the current work directory. Another reason to kept the output None, is to be familiar to output folder name created by the software when savepath is not given or wrong. However, user can create his own temp directory as (e.g. C:/tmp) to save all example outputs.

3. PyCSAMT structure

The toolkit is structured into three (03) main sub-packages with different roles: *ff*, *geodrill* and *viewer* sub-packages. *ff* sub-packages is mainly focused on CSAMT *f*ar-*f*ield data processing and inversion while the *geodrill* sub-package deals with geology data, boreholes and/or wells data. The *viewer* sub-package only deals with visualization (1D and 2D). Overall, the number of sub-packages is estimated at seven (7) which brings basic features in classes, methods and functions. This workbook is structured according to some of main modules in pyCSAMT: *Core*, *processing*. *modeling*, and *geodrill*. Please refer to https://github.com/WEgeophysics/pyCSAMT/wiki/Synopsis-of-pyCSAMT-sub-packages to have more ideas to how the package is arranged.

4. Core

Core sub-package contains modules for basically handling structures within CSAMT data processing (analysis and imaging). The sub-package is the file manager of **.edi*, **avg* and **j* formats (reading, writing and rewriting files) and data provided in different formats are systematically converted into standard units of phase ∅, impedance Z, apparent resistivity (ρ), **E** and **H** fields magnitudes. The inputs data are stored into Python class object and the geolocation data are convoyed to the *cs* module to generate a specific site object that can easily be extracted with attributes.

4.1. Write Zonge **.avg* astatic format(type 2) to the plainly zonge file (type 1)

Brevity, Zonge **.avg* file is a plain file which includes station number, frequency, apparent resistivity, impedance phase, error propagations, generated by the Zonge AMTAVG hardware. **E**- and **H**-fields channel magnitude measured, are calculated and averaged and normalized by current. AMTAVG hardware generates and outputs of station profile (*.stn) format which contain the profile information such as location, easting, northing and elevation value at each offset.

Sometimes the **.avg* file have been pre-processed in the field by applying frequency filters with Zonge ASTATIC software (we call this kind of file, file-type 2). Fortunately, the software could give a possibility to convert the preprocessed (type 2) back to raw plainly **.avg* file (type1) including all errors propagations for data processing

```
In [5]: # import required modules
import os
from pycsamt.ff.core.avg import Avg
#  Define the path to your avg file (type2)
avg_file_2 = os.path.join(os.path.abspath('.'),'data',
'avg','K2.AVG')
# define your savepath
savepath =None      # can be r'C:/temp
# Create an AVG object
avg_obj = Avg(avg_file_2)
```

The avg_obj contains all the data from the zonge avg file, stations names, phase in milliradians, apparent resistivity in ohm.m, E-H magnitudes, E-H-error propagations frequencies, frequency as well as receiver and Transmitter infos (see our git-repo for standard units used). To look for any others parameters information, user can type:

- to get file Header information:

```
# Get Headers information of AVG files
avg_obj.Header.HardwareInfos.version    # version of hardware
avg_obj.Header.SurveyAnnotation.acqdate #  acquisition date
avg_obj.Header.SurveyConfiguration.lineName # name of line
Out[2]: '"LCS01"'
```

- To get Data_section information

```
# For example to get data_section  informations of AVG files :
avg_obj.Data_section.Amps.nfreq         #number of frequency
avg_obj.Data_section.Station.names      # name of stations
avg_obj.Data_section.Resistivity.loc['S00'] # resistivity at 1rst
station
avg_obj.Data_section.Phase.loc['S00'] # phase value at first
station
avg_obj.Data_section.sPhz.loc['S00'] # error phase at station
'S00'
avg_obj.Data_section.pcRho.loc['S00'] # resistivity error
propagation at `S00`
avg_obj.Data_section.Frequency.value   #  to see frequency value
of survey
Out[3]: array([1.000e+00, 1.410e+00, 2.000e+00, 2.810e+00,
4.000e+00, 5.630e+00, 8.000e+00, 1.130e+01, 1.600e+01, 2.250e+01,
3.200e+01, 4.500e+01, 6.400e+01, 9.000e+01, 1.280e+02, 1.800e+02,
2.560e+02, 3.600e+02, 5.120e+02, 7.210e+02, 1.024e+03, 1.441e+03,
2.048e+03, 2.882e+03, 4.096e+03, 5.765e+03, 8.192e+03])
```

Now to convert avg astatic file (type2) to Zonge plainly file (type 1):

```
In [6]: avg_obj.avg_write_2_to_1()

--->  Your <K2.AVG> is rewritten to <K2_2_to_1.avg> successfully. <-
----
```

4.2. Write zonge *.*avg* plainly zonge file to *.*j* (*.*dat*) format

To rewrite zonge *.*avg*, you need to add zonge station profile in *. *stn* format. For instance:

```
1. In [7]:# zonge plainly avg file
2. avg_file = os.path.join('data/avg', 'K1.avg')
3. # zonge station profile location
4. station_profile_file = os.path.join('data/avg', 'K1.stn')
5. #Create avg object
6. avg_obj =Avg(data_fn = avg_file,
7.              profile_fn = station_profile_file)
8. # call function to write j files.
9. # Note: several ways to outputs j-files is available but we use
   default outputs.
10.  avg_obj.avg_to_jfile(savepath =savepath)
```

If the savepath is still None , the default savepath is '_output_J'

```
1. ----------------------------------------------------------------
   ---> 47 J-files have been rewritten.---> see
   path:<C:\Users\Administrator\OneDrive\Python\pyCSAMT\_outputJ_> -
   ----------------------------------------------------------------
```

4.3. Quick analysis with *.*avg* file (E-H magnitude with errors propagations)

Before converting raw data in ***.*edi*** format, user can do a quick analysis of **E-H**-magnitudes with errors propagation to see how raw data as well as error propagations, are arranged at different stations. We intend to visualize station *'S00'*, `S22` and `S46` for this example.

```
1. In [9]: from pycsamt.viewer.plot import Plot1d
2.     ...: Plot1d().plot_curves(fn = avg_file,
   selected_stations=['S00', 'S22', 'S46']  # or [1,23,47]
3.                          )
```

Figure 1 shows an example of quick analysis with data collected with Zonge AMTAVG hardware.

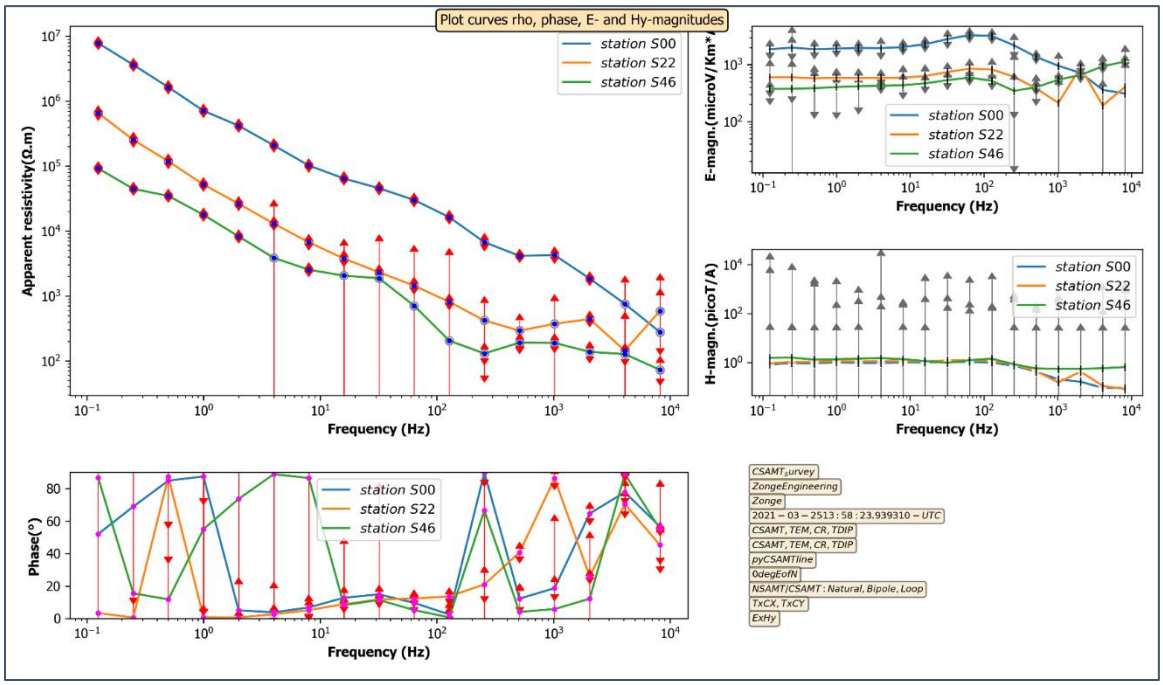

***Figure 1**: Quick analysis proposed with plainly *avg file at three different stations (S00, S22, S46). Errors propagation and coefficients of variations are visualized on error bars plots. The arrow up and down indicated the level of error fluctuation between three stations: from station S00 to S22 and to station S22 to station 46.*

4.4. Convert raw **.avg* file to SEG **.edi* file

It's possible to apply filters when converting raw **.avg* file to SEG **.edi* file, by adding the name of the available filter (e.g. 'tma' , 'ama' or 'flma' filter) through the argument apply_filter. (see https://github.com/WEgeophysics/pyCSAMT/wiki/How-pyCSAMT-works-%3F for details about these filters). The kind of data provided (MT or EMAP data) is detected automatically. The example below is to write plainly *K1.AVG* file with station profile *K1.stn* to EMAP **.edi* format without apply any filter.

Note: The reference_frequency is the frequency at clean data. If it's not given (mean None), it will be automatically computed.

```
1. In [12]: avg_obj.avg_to_edifile(data_fn=avg_file,  # avg file
2.      ...:       profile_fn = station_profile_file, # profile file
3.      ...:       savepath =savepath, reference_frequency= None)
```

The default savepath is _outputAVG2EDI_ if not given.

```
1. ---> 47 Edi-files have been rewritten.---> see
   path:<C:\Users\Administrator\OneDrive\Python\pyCSAMT\_outputAVG2E
   DI_>
```

Note : the new edifies rewritten from zonge *****.*avg* (*K1.AVG*) file and station location profile *****.*stn* (*K1.stn*) file will be used for the following examples.

4.6. Penetration depths 1D & 2D

Penetration depth (1D or 2D) can be visualized a for a better understanding of frequency variations in deep when the magnetic field passes through a conductive or resistive geological structure.

From the penetration 1D plot, different frequencies are visualized on the same graphical. If the input frequency is not in the survey frequency range, the selected frequency will be interpolated.

The example below is to visualize the penetration 1D at 1024Hz, 4096 Hz, and 8000 Hz around *1km* depth.

```
1. In [13]: from pycsamt.viewer.plot import Plot1d
2.          :edipath ='data/edi' # path to edi-files
3.      ...: selected_frequencies =[1024, 4096., 8000.] # selected
   frequencies to visualize
4.      ...: Plot1d().penetration1D(fn =edipath ,
5.      ...: selected_frequency =selected_frequencies)
```

The reference output is illustrated on figure 2 below:

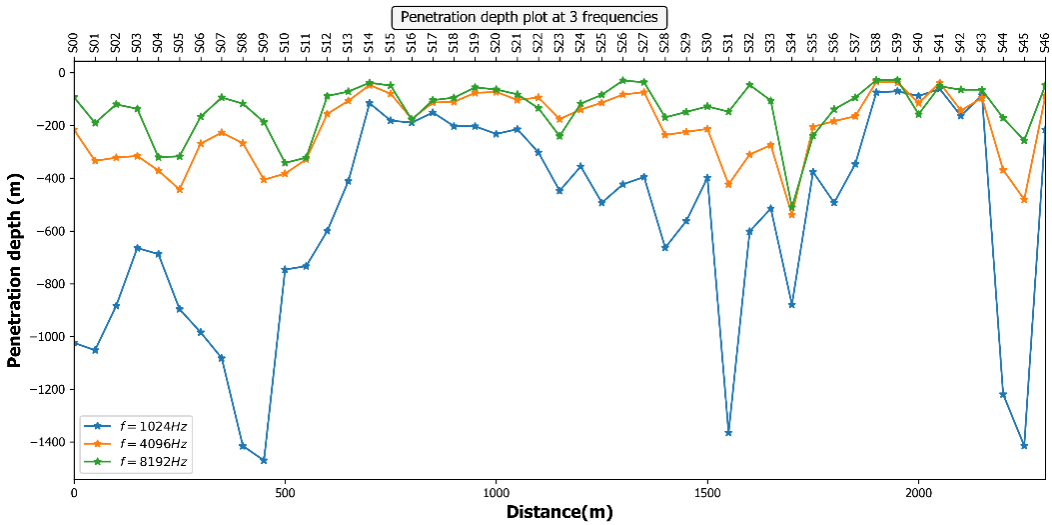

***Figure 2**: 1D Penetration depth plots of Line K1 at three selected frequencies*

We noticed input 8000Hz is not in frequency range then it has been interpolated to 8192 Hz .

To plot penetration 2D , user needs to specify the value of imaging depth using doi parameter (depth of investigation). For instance, to expect to getting more representative frequencies at 1*km* depth for imaging, it is better to set the doi param to *2km* around 2 times the assuming image depth(*1km*).

```
1. In [14]: from pycsamt.viewer.plot import Plot2d
2.      ...: Plot2d().penetration2D(fn = edipath, doi ='2km') # can
   be doi=2000.
```

Figure 3 shows the reference output :

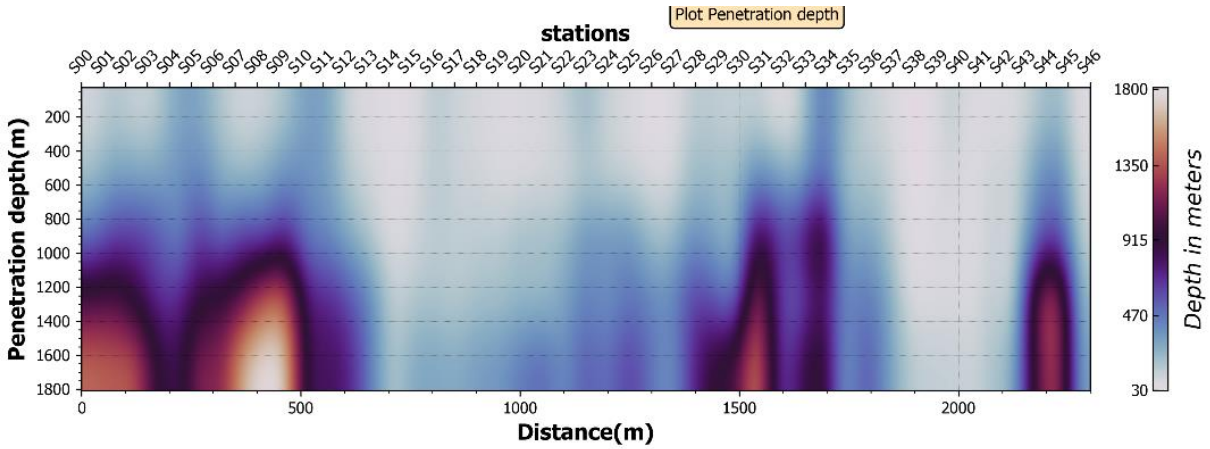

***Figure 3**:2D Penetration depth of Lines K1 at 1.8km depth*

4.7 . Scaled profile

4.7.1. coordinates scaling

Scaling profile is necessary especially when data are collected in hard environment like mountains or an area with many hills or for others reasons. Mathematically, it's sometimes difficult on field to get the exact difference between dipole length at every site. To scale profile and to compute dipole length or to readjust coordinates values for others purposes, , it possible using cs.Profile module of Core sub-package by creating a profile object to hold all information of survey line . For instance:

```
In [15]: from pycsamt.ff.core.cs import Profile
    ...: # create profile_obj
    ...: profile_obj = Profile(station_profile_file)
```

When object profile is created, it is mere to get some relevant attributes like:

```
In [23]: profile_obj.east # get easting coordinate
    ...: profile_obj.elev # get elevation on the survey line
    ...: profile_obj.north # get northing coordinates
    ...: profile_obj.lon # longitude value on survey
    ...: profile_obj.lat # latitude value on survey : see below
Out[23]:
array([26.05230726, 26.05208458, 26.0517252 , 26.05137889,
   26.05108353,26.05074115, 26.0504988 , 26.05032881, 26.05009955,
   26.04981744,26.04956217, 26.04931132, 26.04901795, 26.04869191,
   26.04847115,26.04826015, 26.04798321, 26.04766625, 26.04733678,
   26.04710978,26.0469668 , 26.0467177 , 26.046413 , 26.04622742,
   26.04600258,26.04569437, 26.04538408, 26.04512998, 26.04490923,
   26.04466855,26.04438343, 26.04411091, 26.04385781, 26.04361471,
   26.04335786,26.04309575, 26.04285299, 26.04257855, 26.04230444,
   26.04205242,26.04179832, 26.04153463, 26.04128753, 26.04107128,
   26.04081993,26.04050247, 26.04020718])
```

- To also get the dipole length, user needs to easily type:

```
In [24]: # to get dipole length in meters
    ...: profile_obj.dipole_length
Out[24]: 50.0
```

- To get stations interval , station position and station azimuth :

```
In [26]: profile_obj.stn_interval # interval between stations
    ...: profile_obj.stn_position # scaled position of each
   stations
    ...: profile_obj.azimuth # azimuth of profile_line
Out[26]:
array([108.204, 129.869, 151.533, 149.503, 141.823, 149.146,
   133.186,120.059, 131.239, 140.719, 134.47 , 135.211, 142.707,
   142.788,128.419, 130.516, 140.619, 144.767, 145.258, 130.618,
   116.524,135.118, 141.601, 123.032, 130.778, 141.712, 143.539,
   136.657,128.107, 131.201, 140.53 , 138.713, 134.968, 134.893,
   136.418,135.557, 133.416, 138.651, 141.913, 137.155, 134.354,
   137.384,135.565, 128.107, 133.37 , 143.672, 141.637])
```

To get a single value of specific site(station) of any kind of attribute from Profile object like latitude/ longitude, easting/northing , azimuth or elevation , user must import the required module *Site* like :

```
In [29]: from pycsamt.ff.core.cs import Site
...: site_obj= Site(station_profile_file)
```

```
3. ...: site_obj.east['S00'] # station S00 easting
4. ...: site_obj.north['S00'] # station S00 longitude
5. ...: site_obj.elev['S00'] # elevation of station S00
6. ...: site_obj.lon['S00']
7. ...: site_obj.lat['S00']
8. Out[29]: 26.05230725828611
```

```
1. In [30]: site_obj.azim['S00']
2. Out[30]: 108.204
```

Note: Whatever type of coordinates type is provided, coordinates are systematically converted to both types i.e lat/lon specifying the UTM zone, and easting and northing coordinated values.

Furthermore, to rescale station profile, three (03) modes are available: 'classic(clas)' or natural/distortion(nat/dist) and equidistant(equ) mode. User can either keeps the new scaling station profile (*.stn) rewritten file or user could decide at the same time to plot the both station profiles (scaled and unscaled profile) and keeps the new scaling profile. For this example, we selected the second option to do the both tasks. At the same time, user can also decide to readjust the coordinates if stations real coordinates must be hidden for safety or confidentiality. In that case, user can set the tuple X, Y values into reajust_coordinates parameter. If not given, X=0 and Y=0.

Let's do an example of scaling profile '*K1.stn*':

```
1. In [33]: straighten_out_mode ='classic' # can be
   'natural/distord' or equidistant
2. ...: # contrinute value for x,y coordinates hidden.
3. ...: adjust__x_utm_coordinates = -300238.702
4. ...: adjust__y_utm_coordinates = -2369.252
5. ...: get_new_station_profile =True
6. ...: Plot1d().plot_station_profile(fn = station_profile_file,
7. ...: reajust_coordinates=(adjust__x_utm_coordinates,
8. ...: adjust__y_utm_coordinates),
9. ...: straighten_type =straighten_out_mode ,
10.  ...: outputfile =get_new_station_profile,
11.  ...: savefig=savepath)
```

Setting output-file argument to True, assumes to rewrite a new station profile *.stn . Many others arguments can be provided to control the output files. Refer to documentation for more explanation.

```
1. ----------------------------------------------------------------
2. ---> New <STN-28800_reaj.stn> station file has been rewritten.
3. ---> savepath : <C:\Users\Administrator\OneDrive\Python\pyCSAMT>
4. ----------------------------------------------------------------
```

Note: When object Profile is called, the program automatically scaled the raw station profile using default argument classic. If you don't want to scale the coordinates values, add the word _reaj or _sca or _cor to your profile name : For instance , user does not want to scale the station profile '*K1.stn'* then its new profile name should be : *K1_reaj .stn* or *K1_cor.stn* or *K1_sca.stn.*

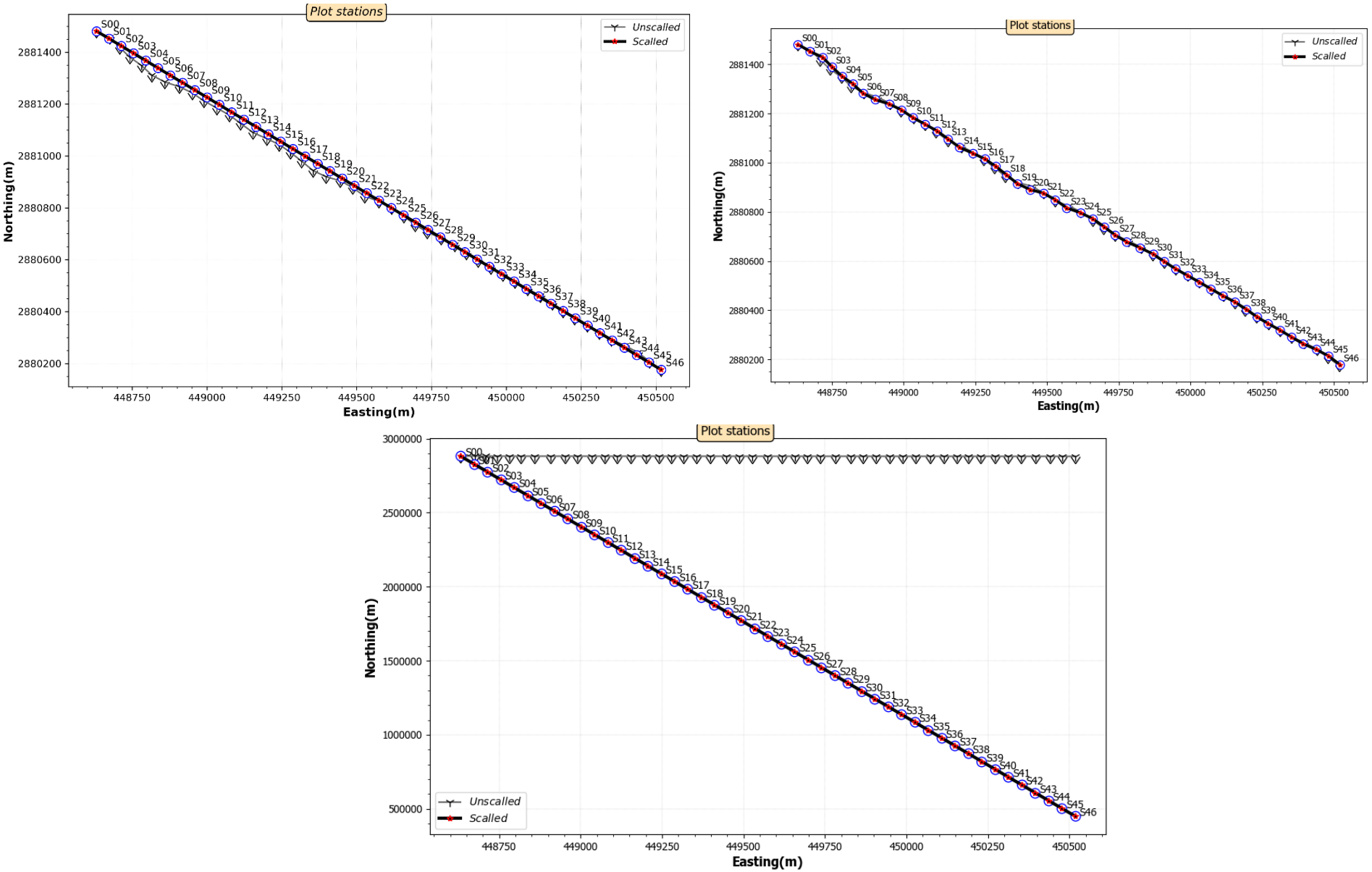

Figure 4: *Three type to scale coordinates: a) classic method, b) natural or distort method, c) equidistant method*

4.7.2. Additional profile features

With *. *edi* files or *.*j-file* or *.*stn* profile file , user can decide to visualize the topography, the station separation profile and azimuth of each survey line on the same graphical or individually using:

```
1. In [3]: from pycsamt.viewer.plot import Plot1d
2.     ...: set_stnNames =True # show staion names labels
3.     ...: plot_type ='*' #or 123 or can be [Top|az|sep] or [1|2|3]
   for individually
4.     ...: path_to_stn_profile_file = 'data/avg/K1.stn'
5.     ...: Plot1d().plot_topo_sep_azim(fn = None, # set edipath or
   jpath
6.     ...: profile_fn= path_to_stn_profile_file ,
7.     ...: plot=plot_type,
8.     ...: set_station_names=set_stnNames,
9.     ...: )
```

Figure 4 shows different plots:

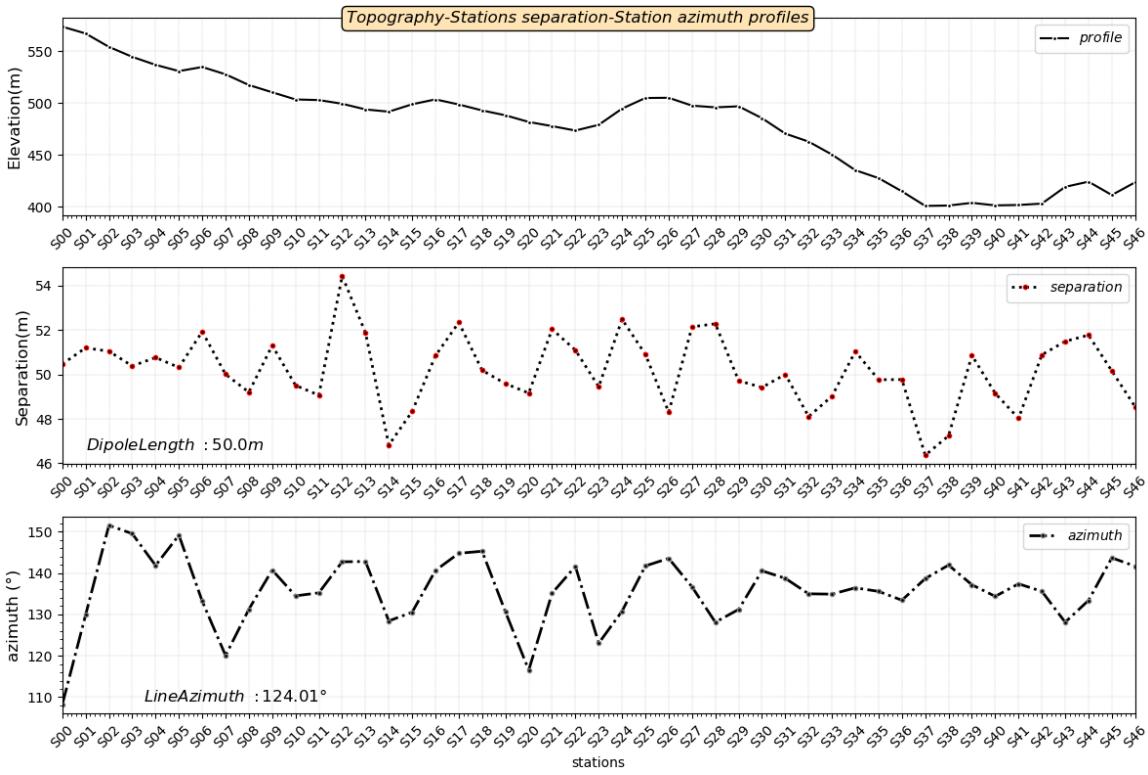

*Figure 5: Topography -separation and azimuth line plot of line K1. Can be used *edi-files and will get the same results*

Additionally, multi-lines plot is also possible using the plot_multiStations function by setting the directory path where profiles files are located to path argument. However, some specific lines can be plotted by setting profile_lines parameters to enumerating a

list of station profile name. For this example, we intend to plot four (04) survey lines (K6.stn to K9.stn) on the same graphical.

```
In [10]: from pycsamt.viewer.plot import Plot1d
...: # path to profiles stn files location
...: path_to_profiles = 'data/stn_profiles'
...: # if you want to plot some specific profiles betwen many
profiles
...: # specify the profile lines Like ['K9.stn', 'K8.stn']
...: profile_lines = ['K9.stn', 'K8.stn'] #profile_lines =
['K9.stn', 'K8.stn']
...: #scaled the line scale # can be `m` or `km` . Default is `m`
...: scale ='km'
...: #save figure
...: savefig ='test_multisites.png'
...: #set to False if you dont want to see stations labels
...: show_station_labels = True
...: Plot1d().plot_multiStations(path = path_to_profiles,
...: profile_lines =profile_lines,
...: scale =scale,
...: savefig =savefig,
...: show_station_labels = show_station_labels )
```

Figure 6 shows the 04 lines plotted on the same graphical with different azimuth lines enumerated:

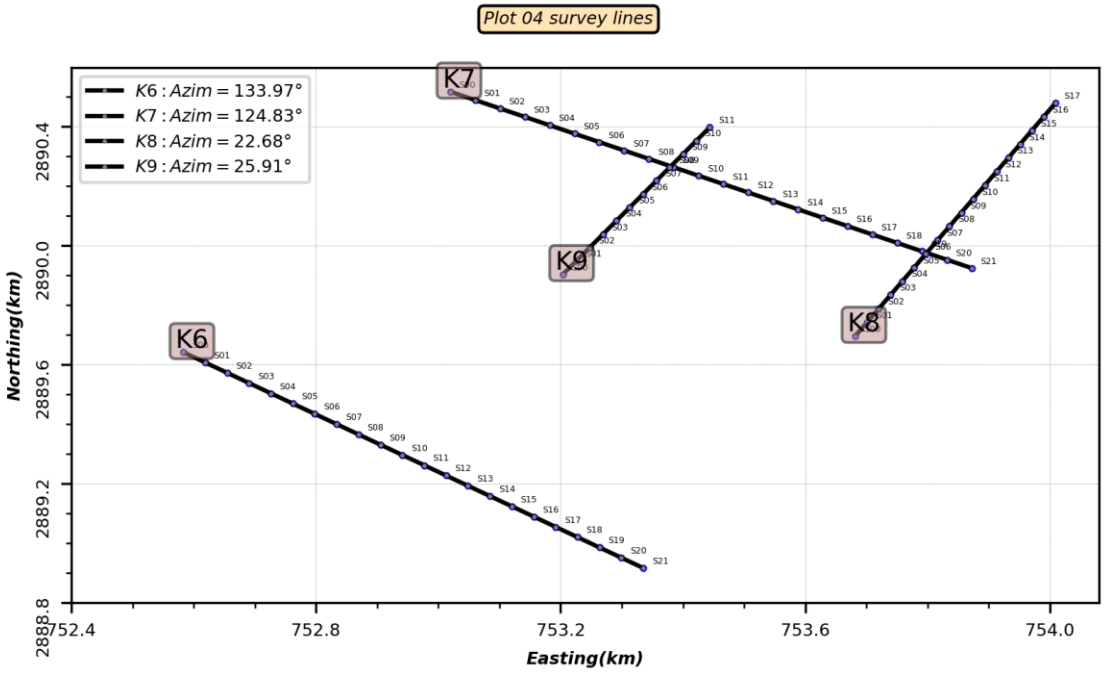

Figure 6: *Plot multi-lines :04 survey lines*

4.8. Pseudo-cross-section of resistivity and phase

The pseudo-section of resistivity and phase illustrate the lateral variation of the resistivity values and phase values through total frequencies and also give an overview of the distribution of resistivity anomalies in the area. Both graphical plots are visualized possible through the following line of code. We noticed that delineate_phase (in degree) and delineate_resistivity (in ohm-meters) are optional parameters.

```
1. In [39]: from pycsamt.viewer.plot import Plot2d
2.      ...: Plot2d().pseudocrossResPhase(fn=edipath,
3.      ...: delineate_pahse =[45],
4.      ...: delineate_resistivity=[1000])
```

The reference output is illustrated in figure 7 below:

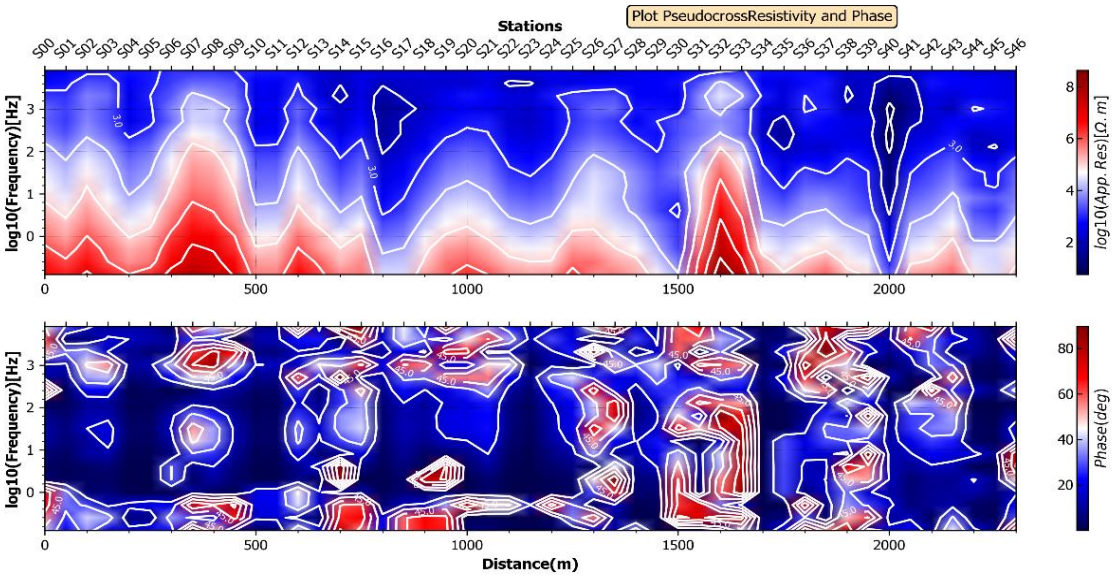

Figure 7:*Pseudo-cross-sections of apparent resistivities and phase maps of CSAMT Line K1. Pseudo-cross sections of apparent resistivities on the top and phase on the bottom.*

5. Processing

5.1. Plot static correction

To plot static correction, the reference frequency is needed as well as the number_of_points param (for *tma*|*flma* filters) and the number_of_skin_depth param (for *ama* filter). For the remainder, available CSAMT filters are *flma*= fixed length-dipole moving average , *tma*=trimming moving-average , *ama*=adaptative moving average. To having control of expected results, it's better to provide dipole length value by setting dipole_length param (in meter). If not provided, dipole length will auto- computed. To

also plot the three filters on the same graphical, user must set ADD_FILTER param with the joker "*". For instance:

```
In [41]: from pycsamt.viewer.plot import Plot1d
     ...: Plot1d().plot_static_correction(ADD_FILTER='*',
     ...: data_fn =edipath,
     ...: frequency_id = None ,
     ...: number_of_points =7. ,
     ...: number_of_skin_depth =7.
     ...: )
```

The following output shows that *reference frequency* is computed automatically if not given.

```
** Reference frequency          = 8196.722 Hz.
** Filter's width               = 627.02 m.
---> Filters TMA & FLMA & AMA are done !
```

Figure 8 shows the three filters application at a single reference frequency to correct apparent resistivity.

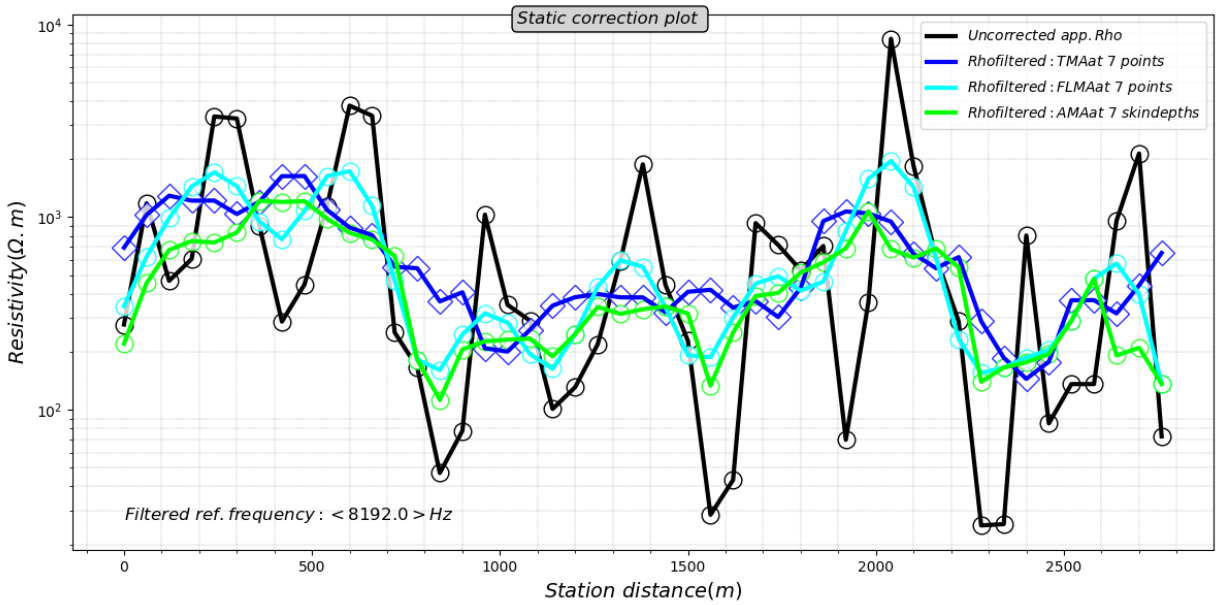

Figure 8: *Profile plots of apparent resistivity at the static-correction-reference frequency show the effects of static-correction moving-average filters. For this data set, a seven skin-depth wide adaptive moving-average (AMA) filter has an average width of 627m, but still creates a slightly rougher profile than either the TMA or FLMA filters. Using a trimmed-moving-average (TMA) filter removes single station offsets, but maximizes sharpness across vertical contacts. A five-dipole-width, fixed length-moving-average (FLMA) filter creates the smoothest profile.*

5.2. Write corrected edi-files

Module pycsamt.ff.processing.corr.shifting.write_corrected_edi is used to correct **.edi* files, and to have control of output files. In addition to *ama*, *flma* and *tma* filters, the software used filters proposed by Krigger and J.R peacob on MTpy software . *ss* (static shift removal) and *dist(distorsion removal*) filters are also suitable for MT data processing edi-files provided are MT data . However, to correct MT edi-files , we absolutely need (reduce_res_factor_x and reduce_res_factor_y params for '*ss*' filters (default =1 if not given) and (distortion_tensor and distortion_err_tensor) params for '*dist*" filters.

For this example, we intend to correct raw **.edi* files from line (K1) by applying the *ama* filter at *7* skin depths with reference frequency equal to *8192*. Hz.

```
In [41]: from pycsamt.ff.processing.corr import shifting
    ...: shifting().write_corrected_edi(data_fn =edipath,
    ...:         reference_frequency=8192.,
    ...:         number_skin_depth=7., # can be 1 to 10
    ...:         dipole_length =50. , # dipole length in meter
    ...:         FILTER ='ama',
    ...:         savepath=savepath) # save output corrected files
```

The default save path if not provided is _outputEDI_

```
------------------------------------------------------------------
---> Edifile <new_new_csc300.edi> has be successfully written to
your savepath.
---> savepath :
<C:\Users\Administrator\OneDrive\Python\pyCSAMT\_outputEDI_>
--> ! Filter `ama` is successfull done !
** Filter                      = AMA
** Type of correction           = Adaptative moving-average
** Number of skin depth         = 3
** Dipole length                = 50.0 m.
** Reference frequency          = 8192.0 Hz.
 ** Frequency numbers             = 17
 ** Number of sites processed    = 47
 --> corrected edi  successfully done !
 ------------------------------------------------------------------
```

5.3. Plot and compare multiple corrections

The better way to visualize the implementation of different filters is to create a loop of different corrected edi-path and to compare the expected results from uncorrected resistivities. For this example, we created a loop to only visualizing the pseudo-cross-section of apparent resistivity. Figure 09 illustrates an example of pseudo-cross-section of uncorrected apparent resistivities and the corrected resistivities using filters *tma*, *flma*, *ama*, of line K1. Before using this loop, user needs to write corrected edi from each filter above.

```
1. In [42]: from pycsamt.viewer.plot import Plot2d
2.      ...: contouRes = 1000. # resistivity in ohm.meters
3.      ...: for path2edi_obj in [
4.      ...: 'data/edi',
5.      ...: 'data/correctedEDI_TMA', # edi corrected with tma filter
6.      ...: 'data/correctedEDI_FLMA', # edi corrected with flma
7.      ...: 'data/correctedEDI_AMA', # edi corrected with ama
8.      ...: ]:
9.      ...: Plot2d().pseudocrossResPhase(fn=path2edi_obj,
10.  ...: delineate_resistivity=[1000])
```

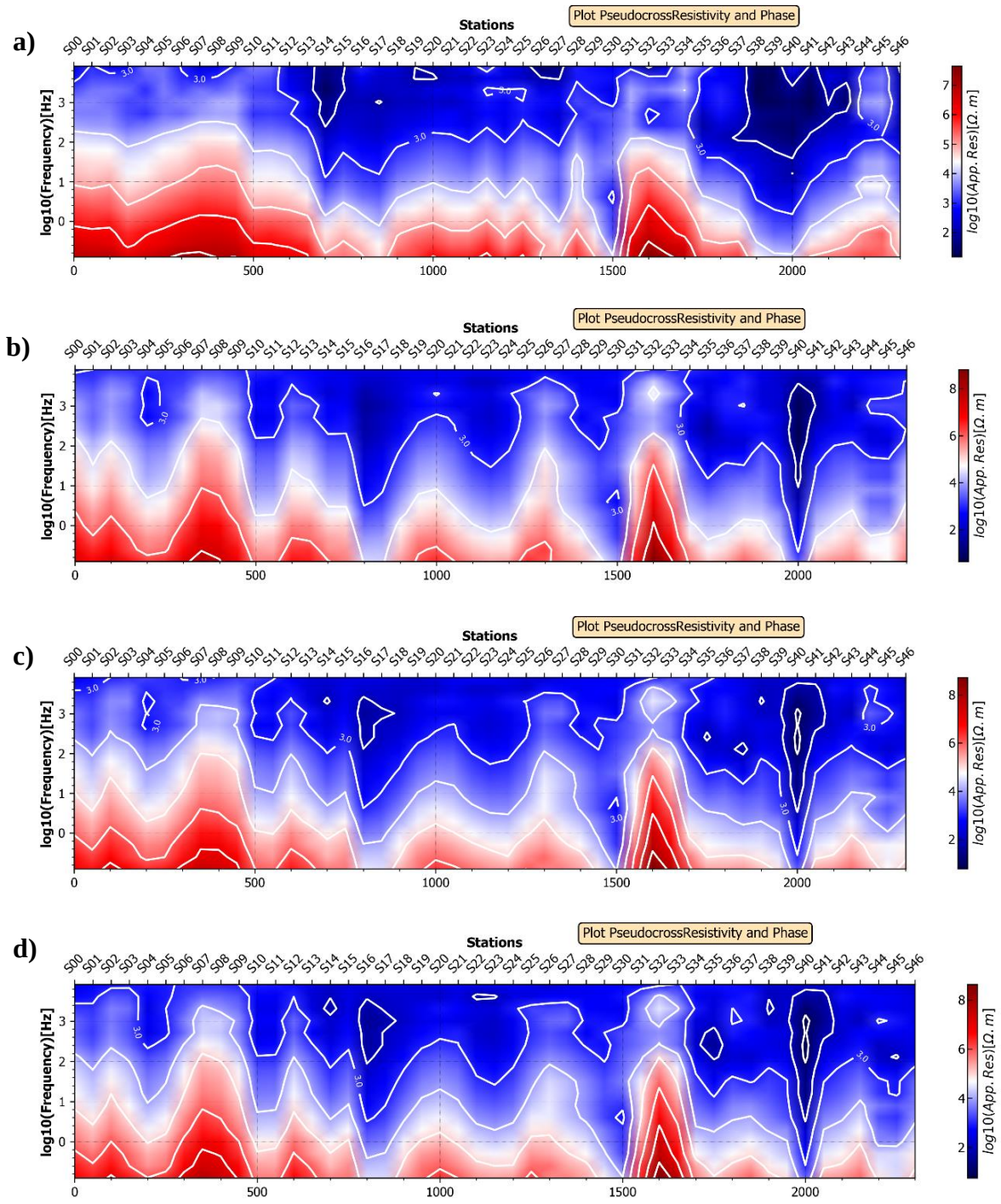

Figure 9: Sample test of filters application on pseudo-cross-section of -resistivity of line K1. Filters tma, flma, and ama, are applied to corrected apparent resistivities. a) Uncorrected apparent resistivities map. b), c) and d) pseudo-cross resistivity corrected maps with TMA, FLMA filter and AMA filters respectively

6. Modeling with occam2D

6.1. Build Occam2D input-files

The software is able to can call MTpy toolbox to build *occam2d input files* from S*.edi files when using modeling sub-package. However, if MTpy is not installed yet, it's feasible to forthright run pycsamt.modeling.occam2d.occam2d_write.buildingInputfiles to automatically install the software and its dependencies. However we recommend to avoid conflict of packages to set a new virtual environment (see software installation guide). Occam2d input files (*Occam2DMesh*, *Occam2DModel*, *Startup'*, *'OccamDataFile.dat'*) can be built by entering input parameters as below:

```
In [45]: from pycsamt.modeling.occam2d import occam2d_write
...: occam2d_write.buildingInputfiles(edi_fn =edipath,
...: geoelectric_strike= 34., # If None , should be 0.
...: interpolate_freq= True, # interpolate if you want
...: intp_freq_logspace =(-1, 4, 17),
...: iteration_to_run= 100., # number of iterations
...: resistivity_start = 1., # in log10 resistivity
...: res_tm_err= 10., # in %
...: phase_tm_err= 20. , # in %
...: occam_mode= '6', # see documentation
...: n_layers =31. ,  # number of model layers
...: cell_width = 5 ,  #
...: x_pad_multiplier = 1.7,
...: trigger =1.12,
...: z_bottom =5000.,  # exaggerated depth bottom
...: z1_layer =5.,     # first layer thickness
...: z_target = 1100., # imaging depth (vertical z )
   ...: )
```

For demonstration, we specified all parameter values to have control of what we expect to get. However, for the CSAMT survey, some parameters are defaults and optional. Refer to documentation for more details. The default savepath is 'occam2dBuildInputfiles"

```
Wrote Mesh file to
C:\Users\Administrator\OneDrive\Python\pyCSAMT\occam2dBuildInputf
iles\Occam2DMesh
Wrote Regularization file to
C:\Users\Administrator\OneDrive\Python\pyCSAMT\occam2dBuildInputf
iles\Occam2DModel
---> Build occam2d Regularization mesh done !
Wrote Occam2D startup file to
C:\Users\Administrator\OneDrive\Python\pyCSAMT\occam2dBuildInputf
iles\Startup
---> Build write occam2D startup file done !
--------------------Summary *occam2d input params* infos---------
** Given frequency number = None
```

```
8. ** Interpolate frequencies range = (-1, 4, 17)
9. ** Occam model mode = 6
10.  ** TM rho error floors = 10.0 %.
11.  ** TM phase error floors = 20.0 %.
12.  ** Model cell width = 5
13.  ** model horizontal pad = 1.7
14.  ** Model bricks trigger = 1.12
15.  ** Number of model layers = 31
16.  ** Top layer thickness = 5.0 m.
17.  ** Expected image depth = 1100.0 m.
18.  ** Model bottom = 5000.0 m.
19.  ** Expected iteration to run = 100.0
20.  ** starting model resistivity = 10.0 ohm.m
21.  ** Geoelectric strike = +34.0 degrees E of N
22.  ------------------------------------------------------------
   ---> Building occamInputfiles function successfully run. !
```

6.2. Plot RMS via occam2d log file

Actually pyCSAMT cannot straightforwardly call OCCAM2D software to invert CSAMT data. However, we can get occam2d software though: https://marineemlab.ucsd.edu/Projects/Occam/index.html and compile it afterwards to run it into your default platform (Linux or Window). When inversion files are created OCCAM 2D will provide iterations files (*.iter) and response (*.resp) files and logfiles(*.logfile).

Before plotting the expected model, it's better to select among different iterations which model fits the best one. Moreover, the OCCAM2D logfile mostly named '*logFile.logfile' can* be used to plot all iterations and roughness values self-contained in order to choose the preference model after inversion. Furthermore, the OCCAM2D preferred models are right chosen by combining the values of the root mean square (RMS) error and roughness, i.e. the preferred results must be the smallest roughness value at lower RMS level. Such model consequently avoids unnecessary structures but preserve higher resolution.

The following line of codes below helps to implement the building RMS plot on figure 10.

```
1. In [51]: from pycsamt.viewer.plot import Plot1d
2. ...: RMS_target =1. # Root Mean-square target , default is 1.0
3. ...: show_grid =False #set to let grid to be visible
4. ...: showTargetLine= True # show rms target line .
5. ...: savefigure = os.path.abspath('./test_rmsplot.png') #
   savepath
6. ...: Plot1d().plotRMS(fn ='data/occam2D/logFile.logfile',
7. ...:                  target=RMS_target ,
8. ...:                  show_grid =show_grid,
9. ...:                  show_target_line = showTargetLine,
10.  ...:                savefig =savefigure )
```

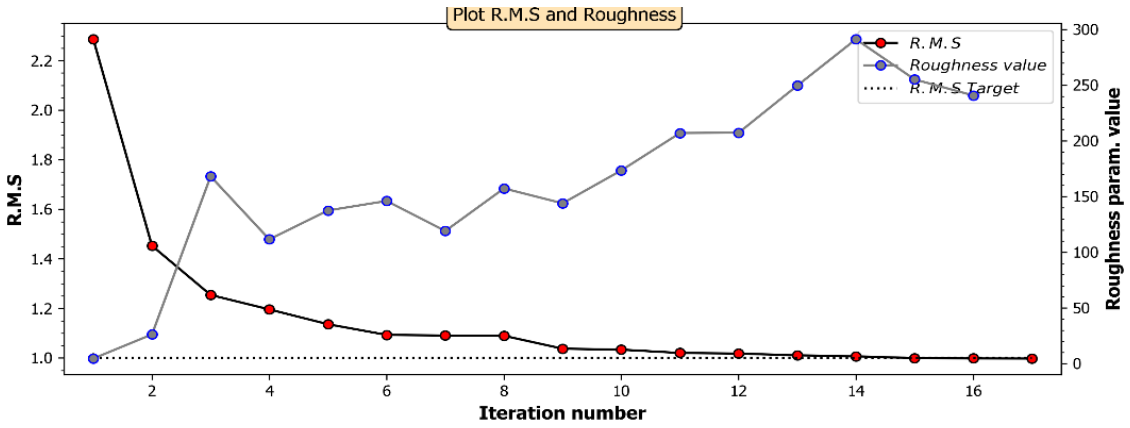

Figure 10: Example of model's selections. Expected RMS target is set to 1.0. The RMS at iteration 17 with low value of roughness is preferred than the iteration 14 although the target RMS is reached. Indeed, the criterion to select the best model does not depend only with the RMS, the OCCAM2D roughness parameter can also influence the choice of model selection.

Note: For the following example we will use *ITER17.iter* as acceptable model.

6.3. Plot occam2D model

For the following demonstration we put occam2d inversion files into *kwargs* arguments (`oc2d_inversion_kwargs`) to avoid redundancy in codes .

```
In [49]: oc2d_inversion_kwargs={
             'data_fn':'OccamDataFile.dat',# occam2d data file
...:         'mesh_fn': 'Occam2DMesh', # occam2d mesh file
...:         'model_fn':'Occam2DModel' , # occam2d model file
...:          'iter_fn': 'ITER17.iter',  #occam2d iteration file
...:           }
...: # join the occam2d directory with occam inversion files
...: # to create a realpath for each file.
...: for oc2dkey, inversion_file in
oc2d_inversion_kwargs.items():
 ...: oc2d_inversion_kwargs[oc2dkey]=
os.path.join('data/occam2D', inversion_file)
```

For this example, we want to visualize a 2D resistivity model at `doi` = 1km depth. Let's go ahead for the following's scripts

```
In [50]: from pycsamt.viewer.plot import Plot2d
    ...: Plot2d().plot_occam2dModel(**oc2d_inversion_kwargs,
    ...: doi ='1km', # depth of investigation
    ...: show_report =True ) #Ground water report
```

if show_report param is set to True, the following report will be generated and may use in groundwater exploration

```
----------------------------Occam 2D Mesh params ----------------
*** Horizontal nodes read = 576 instead of 577 in mesh files.
2021-04-17T 13:55:11 PM - Mesh - INFO - *** Horizontal nodes read = 576 instead of 577 in mesh files.
*** Vertical nodes read = 31 instead of 32 in mesh files.
2021-04-17T 13:55:11 PM - Mesh - INFO - *** Vertical nodes read = 31 instead of 32 in mesh files.
---> Horizontal nodes = 576
---> Vertical nodes = 31
2021-04-17T 13:55:11 PM - Model - INFO - Read Model Blocks and put model resistivity into aggregated mesh blocks
---Boundaries X (Horizontal nodes)---
 ** Minimum offset (m) = -389.0
 ** Maximum offset (m) = 2970.2999999999997
 ----Boundaries Z (Vertical nodes)----
 ** Minimum depth (m) = 0.0
 ** Maximum depth (m) = 5999.0
 ----------------------------Occam 2D Models params---------------
 ** Model layer num. = 26
 ** Model param count = 3752.0
 ** Iteration num. = 17.0
 ** Occam Misfit value = 0.9977012
 ** Occam Misfit reached = 1.0
 ** Occam Misfit target = 1.0
 ** Occam Roughness params = 241
 2021-04-17T 13:55:11 PM - Data - INFO - Read Occam2D data file :<data/occam2D\OccamDataFile.dat>
 2021-04-17T 13:55:11 PM - DataBlock - INFO - Ckeck dataBlocks and  corresponding data params have been set properly.
 ----------------------------Occam 2D Data infos----------------
 ** Sites num. = 47
 ** Frequencies num. = 17
 ** Highest frequency (Hz) = 10000.0
 ** Lowest frequency (Hz) = 10.0
 ** Minimum offset (m) = 0.0
 ** Maximum offset (m) = 2300.0
 ----------------------------------------------------------------
 2021-04-17T 13:55:11 PM - Plot2d - INFO - Ready to plot Model with matplotlib "pcolormesh" style.
 ---> Average Rho on survey area is = 663.7945899726384 Ω.m.
 ---> Probably very conductive zone is = S40 with rho = 78.36826915012188 Ω.m.
 ---> Probably very resistive zone is = S31 with rho = 4045.178160335858 Ω.m.
 - However :
 ---> Minimum ratio is = 0.6711969.
 ---> Maximum ratio is = 1.2781341.
```

and the result of model file after codes implementations is:

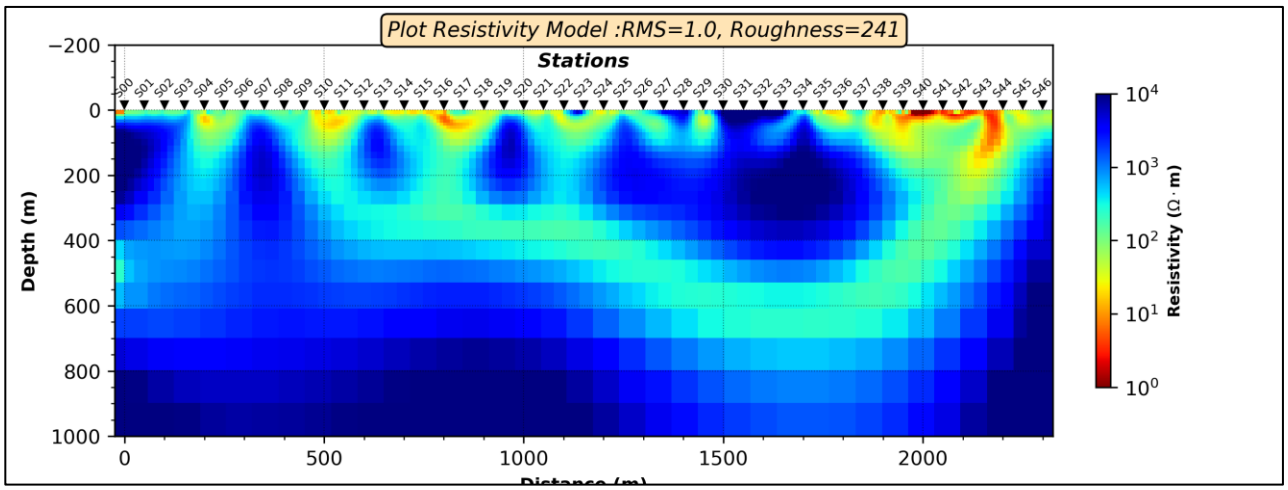

Figure 11: CSAMT 2D inversion models results with errors floors set at 10% apparent resistivities and 20% phase for line K1 with starting model set at 10 Ω.m;

6.4. Write model x,y,z files or Bo yang file

The name of Bo Yang file or model x, y, z, is inspired of MATLAB Iter2dat file, combined with MATLAB plotting tools written by Kerry Key (Scripps Institution of Oceanography Marine EM Laboratory | Research | Occam 2D MT Inversion (ucsd.edu). The codes have been rewritten in Python as Iter2Dat python module, by adding others improvement tools. However, the original MATLAB code is located on 'add.info" directory of pyCSAMT package. x, y, z output resistivity model can be used by suitable modeling external software to plot a 2D resistivity model.

To create x,y,z resistivity model file from OCCAM2D inversion files , we need to import the required module Iter2Dat and follow the script below:

```
In [54]: from pycsamt.modeling.occam2d import Iter2Dat i2d
...:# give an output file name, if None, will create
automatically
...: outputfilename ='testi2d_area'
...: # scale the output data
...: scale_output =None # if None, default is "km".can be "m"
...: # imaging depth : Maximum depth investigation
...: doi = '1km' # can be float like 1000 = 1km
         # provided elevation on list or array_like.
...:  elevation =None
 ...: # create i2d or modelxyz object and entering aruments
 ...: occam_iter2dat_obj =i2d(**oc2d_inversion_kwargs,
 ...: savepath =savepath)
 ...: occam_iter2dat_obj.write_iter2dat_file(filename =
outputfilename,
      ...: scale=scale_output,
```

```
15.      ...: doi=doi,
16.      ...: elevation=elevation)
```

If the savepath is not given, default savepath is created and named _iter2dat_

```
1. 2021-04-17T 15:08:08 PM - Iter2Dat - INFO - Writing a iter2dat
   file from specifics objects
2. ---> files testi2d_area.dat & testi2d_area.bln have been
   successfully written to
   <C:\Users\Administrator\OneDrive\Python\pyCSAMT\_iter2dat_>.
```

6.5. Plot forward response and Residual

After inversion, user can fast-forward plot the forward model and residual model to confirm existence of anomalies. The following output is just an example to implement the code by it does not reflect the effectiveness of data in that area.

```
1. In [79]: contourRes =None # res value in ohm m, can be 1000.
2.      ...: showReport = True
3.      ...: savefigure = os.path.abspath('./test_fwdresp.png')
4.      ...: Plot2d().plot_Response(
5.           data_fn ='data/occam2D/OccamDataFile.dat',
6.      ...: response_fn = 'data/occam2D/RESP17.resp' ,
7.      ...: delineate_resistivity =contourRes ,
8.      ...: show_report =showReport ,
9.      ...: savefig =savefigure
10.           )
```

The following report can be generated:

```
1. ----------------------------Occam 2D Data infos-----------------
2. ** Sites num. = 47
3. ** Frequencies num. = 17
4. ** Highest frequency (Hz) = 10000.0
5. ** Lowest frequency (Hz) = 10.0
6. ** Minimum offset (m) = 0.0
7. ** Maximum offset (m) = 2300.0
8. ------------------------------------------------------------------
9. 2021-04-17T 17:00:16 PM - Response - INFO - Read Occam 2D
   response file <data/occam2D/RESP17.resp>
10.  2021-04-17T 17:00:16 PM - pycsamt.etc.infos - INFO - Reading
   RESP17.resp file
11.  -------------------------Occam 2D Response infos---------------
12.  ** Occam data type = ('tm_log10', 'tm_phase')
13.  ** Occam data mode = (5, 6)
14.  ** Forward shape:TM log10 = (16, 47)
15.  ** Residual shape:TM log10 = (16, 47)
16.  ** Forward shape:TM phase = (16, 47)
17.  ** Residual shape:TM phase = (16, 47)
18.  ====================Occam Response plot infos=============
```

```
19.  ---> Occam 2D plot Mode = TM log10
20.  ---> Occam 2D plot style = "imshow"
21.  2021-04-17T 17:00:16 PM - Plot2d - INFO - Ready to plot forward
   with matplotlib "imshow"
22.  ---> Average Rho on survey area is = 503.5897897389726 Ω.m.
23.  ---> Probably very conductive zone is = S40 with rho =
   0.07672753570837182 Ω.m.
24.  ---> Probably very resistive zone is = S07 with rho =
   10481.688524611265 Ω.m.
25.  - However :
26.  ---> Minimum ratio is = -0.4126636.
27.  ---> Maximum ratio is = 1.4879041.
```

After implementing the code, figure 12 displays an overview of forward model in top and residual model in bottom of survey line.

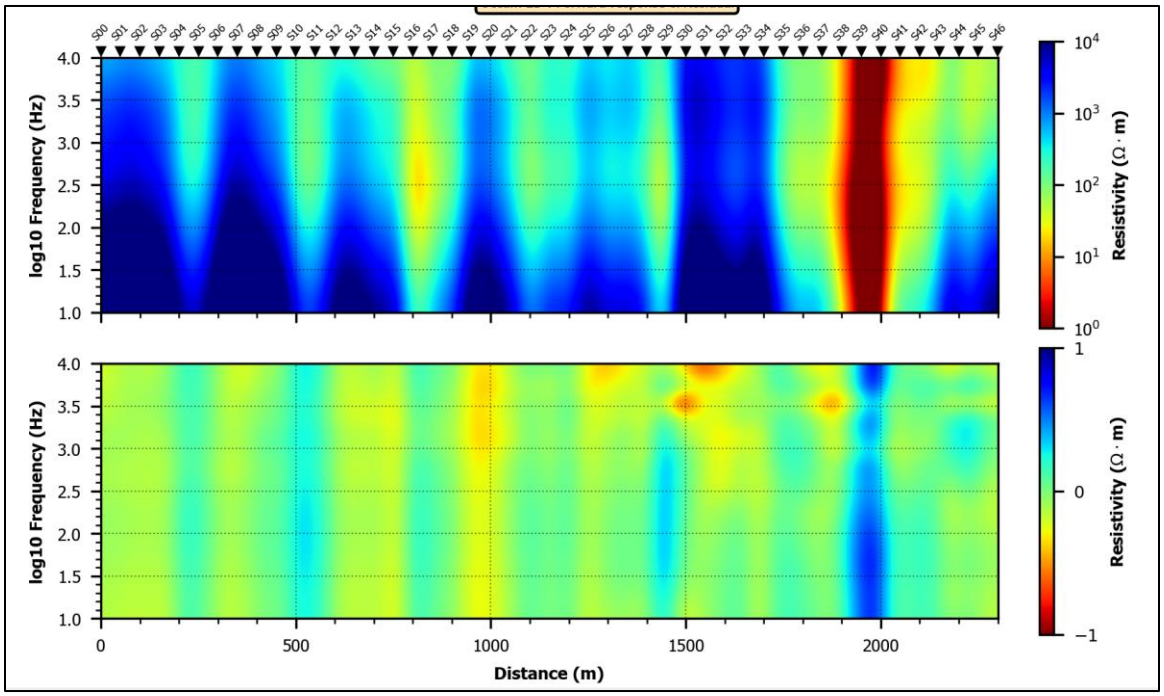

Figure 12: of Line K1 Forward response model (on top) and residual (on bottom)

7. GeoCore

geoCore sub-package mainly deals with geological data (*name of rocks, electrical properties of rocks*). The sub-package also intended to rebuild and plot borehole data obtained from investigation area through a parser file in **.csv* format. The resistivity 2D model from OCCAM2D inversion results can be transformed into '*pseudo- stratigraphy log*' with different layer names by adding *truth resistivity (*input_resistivities*)* values got

on the field to emphasize real underground geological structures. Additionally, *geoCore* generates new outputs such as a *new resistivity model* (_rr model) and *residual model(*_sd model*)* obtained by forcing the program to find close *truth resistivity* model from truth *resistivity* values inputted. Outputs resistivity models can be visualized by *'Golden software'* and/or *'Oasis montaj'* of Geosoft corporation.

One of the main parameters to enhance a geophysical interpretation is "step_descent" parameter. Indeed, to create a new resistivity model, the program divides the input OCCAM2D resistivity model into different resistivities block models at different depth levels. The depth level where the program is assumed to break and to force (average + replaced + roughness) existing value to be close to *input true resistivity* values is called *step decent*. The *step descent* is a variation depth value i.e. the width of broken model block in meters. If *step descent* value trends to be 0. m, no difference will be found between new resistivity model and OCCAM2D inversion model. Also, if *step descent* value is too large, the new model trends to be far from the reality. However, the default threshold is ***20%*** of imaging depth. For instance, if image depth is *1km* then step_descent should be *200 m*. Consequently , to build or to output a new resistivity model file, a step_descent parameter is obviously needed.

7.1. Build pseudo-stratigraphy log

Generating a new resistivity model depend on specific values of geological and borehole/well information collected on survey area such as *true resistivity*, *rocks'* names, *their electrical properties* as well as some *layer names*. True resistivity values (in Ω.m) and the layer names got from the hydrogeological firm and geological company respectively can be included into 2D inversion results. If layer names are not provided, the software will give a corresponding layer names according to standard electrical rock properties contained in inner database. For instance, we assume the following data *river water'(66,70Ω.m), 'fracture zone'(70-180 Ω.m), 'granite'(180-1000.Ω.m), Most weathered granite'(1000.-3000 Ω.m)* and *'less weathered granite'(3000.-1000 Ω.m)* are additional data collected during geophysical survey.

To create a new model, the best tip is to input additional geological information's (additional_geological_infos). into *kwargs* argument:

```
In [80]: # additional geological information collected
...: INPUT_RESISTIVITIES = [66.,70., 180.,
...: 1000., 3000., 10000., 20000.]
...: INPUT_LAYERS = ['river zone', 'fracture zone' ,
...:                 'granite ', 'Most Weathered',
...:                 'Less Weathered']
...: STEP_DESCENT =200. # step descent in meters
...: DOI ='1km' # investigation depath
...: additional_geological_infos={
```

```
...:             'doi':DOI,
...:             'step_descent': STEP_DESCENT,
...:             'input_resistivities' : INPUT_RESISTIVITIES,
...:         'input_layers' : INPUT_LAYERS
...: }
```

For demonstration, we intend to visualize at *1km* depth the stratigraphy log at station *43* . Furthermore, to plot pseudo-stratigraphy as well as to write new outputs models, it's possible to use either OCCAM inversion results (or x,y,z models outputs by setting xyz input files into kwargs arguments (i2d_files_kwargs) . Both will yield the same result. For instance, to plot pseudo-stratigraphy:

- using xyz model or Bo yang file :

```
In [84]: i2d_files_kwargs={
    ...: 'iter2dat_fn' : 'data/iter2dat/K1.iter.dat',
    ...: 'bln_fn':'data/iter2dat/K1.bln'
    ...: }
    ...: Plot2d().plot_Pseudolog(station_id = 'S43',
    ...:                **additional_geological_infos,
    ...:                 **i2d_files_kwargs)
```

- Plot using occam2d inversion files put in *kwargs* arguments.

```
In [81]: from pycsamt.viewer.plot import Plot2d
    ...: Plot2d().plot_Pseudolog(station_id = 'S43', # station to
    ...: **additional_geological_infos,
    ...: **oc2d_inversion_kwargs)
```

And after successfully run, we get the following report.

```
---> resetting model doi to = 999.0 m depth !
** Layers sliced = 7
** Rho range = (66.0, 70.0, 180.0, 1000.0, 3000.0, 10000.0,
20000.0) (Ω.m)
** Minimum rho = 66.0 Ω.m
** Maximum rho = 20000.0 Ω.m
---> !We added other 2 geological strutures. You may ignore it.
2021-04-17T 17:42:37 PM - Geodrill - INFO - Build the
pseudosequences of strata.
** QC flux rate = 75.0 %
** Number of layers = 7
 ---------------------------------------------------------------
 Structure Rho mean value (Ω.m) Rho range (Ω.m)
 ---------------------------------------------------------------
 river zone 68.0 66.0-70.0
 fracture zone 123.0 66.0-180.0
 granite 535.0 70.0-1000.0
 Most Weathered 1590.0 180.0-3000.0
```

```
17.  Less Weathered 5500.0 1000.0-10000.0
18.  Igneous rocks 11500.0 3000.0-20000.0
19.  Igneous rocks 15000.0 10000.0-20000.0
20.  ----------------------------------------------------------------
21.  2021-04-17T 17:42:37 PM - Plot2d - INFO - Build Pseudo drill
   from Occam 2D models.
22.  2021-04-17T 17:42:37 PM - Plot2d - INFO - Ready to plot
   Pseudodrill with matplotlib "pcolormesh"
23.  2021-04-17T 17:42:37 PM - Plot2d - INFO - Build Plot1D
   resistivities sounding curves.
24.  2021-04-17T 17:42:37 PM - Plot2d - INFO - Build Pseudo sequences
   with delais logs sequences curve and average curves..
25.  2021-04-17T 17:42:37 PM - Geo_formation - INFO - Read & and
   decodes geostructures files .
26.  ---> !We added other 2 geological strutures. You may ignore it.
27.  ----> Successfull connexion to geoDataBase !
```

As we can see the report, one layers is topped "*Igneous Rocks*". This occurs because some resistivities values in raw models does not correspond the specific layers names listed on input_layers parameter, then the program searches into the geo-database to find the layers that fit the best electrical property of these resistivities trends.

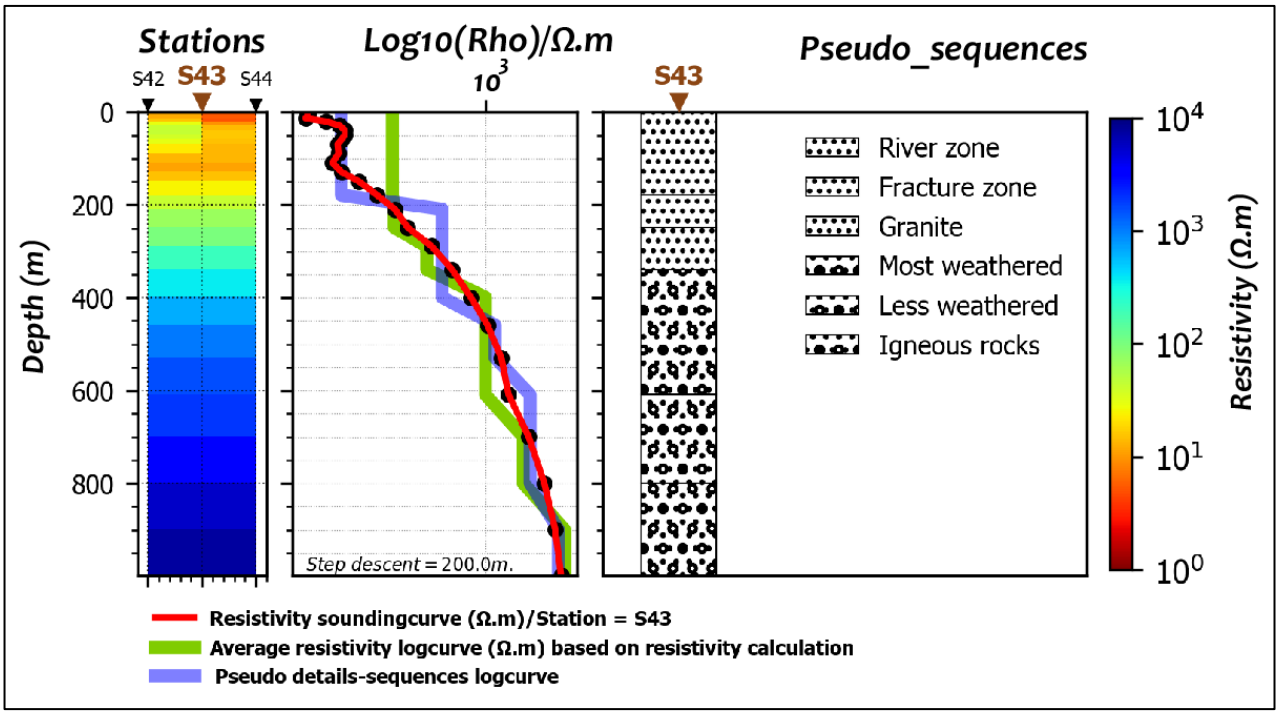

***Figure 13**: Pseudo-stratigraphy log construction from OCCAM2D inversion results at station S43 of line K1. This is a matplotlib representation which does not include FDGC map symbolization. Different zones are demarcated with different granites differentiations. However, we can change the name of Most weathered and Less weathered for the exact name of granites met on survey area.*

Note: If input resistivity values are not given, the program auto-works to find itself input resistivities. It analyses and sliced the resistivities according the existing gap between raw model resistivity and the time for computing little bit delays. Be cautious for automatic model generating because it does not much exactly the underground model. To have full control of output and to get pseudo-obvious model, you need ABSOLUTELY to provide at least the input resistivities.

7.2. Export stratigraphy model files

New geological model can be exported to different external modeling software. The most common output read by most modeling software is the golden output. The software outputs *04* types of files which include the different steps of modeling. The first step is average rho model file (*_aver.dat) and the second step is roughness model file where the program finds a corresponding resistivity, roughs values and replaces it to the truth resistivity(*_rr.dat). The last one is the final true model. Besides this model, there is another model called the *step descent* model (*_sd.dat) . In fact, this model is usefully to know the different zone where the program find difficulties to find the existing equivalent underground structures. Most of case, *step descent* model shows the effectiveness of existing anomalies like disconformities (fractures, dykes, faults and else). The final output is location output in '.bln file(_yb.stn). For good visualization and to see the difference between the yielded models, it's convenient to plot all individually.

Following on our same example, let import the required module Geodrill and try to export the new resistivity files to the different formats before creating a geological self-container object:

```
1. In [86]: from pycsamt.geodrill.geoCore.geodrill import Geodrill
2. ...: # create a geological object from geodrill module
3. ...: geo_obj = Geodrill(**oc2d_inversion_kwargs ,
   **additional_geological_infos)
```

7.2.1. Output to golden software

As we aforementioned, the output resistivity model read by modeling software is golden software outputs. Let's try to outputs the fourth (04) models' files enumerated above.

```
4. In [87]: geo_obj.to_golden_software(filename= 'test_area',
5. ...:                 to_negative_depth =True , # default output
6. ...:                 savepath=savepath)
```

The following output is yielded and the savepath is _outputGeoSD_

```
1. ------------------------------------------------------------------
2. ---> resetting model doi to = 999.0 m depth !
3. ** Layers sliced = 7
4. ** Rho range = (66.0, 70.0, 180.0, 1000.0, 3000.0, 10000.0,
   20000.0) (Ω.m)
5. ** Minimum rho = 66.0 Ω.m
6. ** Maximum rho = 20000.0 Ω.m
7. ---> !We added other 2 geological structures. You may ignore it.
8. 2021-04-17T 19:59:17 PM - Geodrill - INFO - Build the
   pseudosequences of strata.
9. ** QC flux rate = 75.0 %
10.  ** Number of layers = 7
11.  ------------------------------------------------------------
12.  Structure Rho mean value (Ω.m) Rho range (Ω.m)
13.  ------------------------------------------------------------
   river zone 68.0 66.0-70.0
14.  fracture zone 123.0 66.0-180.0
15.  granite 535.0 70.0-1000.0
16.  Most Weathered 1590.0 180.0-3000.0
17.  Less Weathered 5500.0 1000.0-10000.0
18.  Igneous rocks 11500.0 3000.0-20000.0
19.  Igneous rocks 15000.0 10000.0-20000.0
20.  ------------------------------------------------------------
21.  ---> geo output files test_area.4_aver.dat, test_area.4_rr.dat,
   test_area.4_sd.dat & test_area.4_yb.bln have been successfully
   written to
   <C:\Users\Administrator\OneDrive\Python\pyCSAMT\_outputGeoSD_>.
```

7.2.2. Output to oasis montaj

Oasis montaj output is little complex and very interesting for 3D map especially when multiples lines are carried out on survey area. The outputs on excel sheet in ***.xlsx** format or ***.csv** format and the default format is in ***.xlsx** . when the output format is ***.xlsx** , the resulting models are outputs in the main workbook which different sheet correspond to the model generated. (_sd, _rr, or _aver). The file can be export to oasis montaj of Geosoft software to visualize the expected model.

For consistently, Oasis montaj output needs absolutely the station profile which refers to coordinates location of each sites. However, if station profile does not exist, user can build ***.stn** file using Profile module to create one. It's also possible to provide arrays of easting and northing coordinates which perfectly match the number of investigated sites.

```
1. In [87]: geo_obj.to_oasis_montaj(profile_fn ='data/avg/K1.stn',
2. ...: to_negative_depth =True , # default output
3. ...: to_log10=True, #output resistivity to log10 values
4. ...: filename ='test_area',
5. ...: savepath=savepath)
```

The default savepath is _output2Oasis_ and the reference output is below:

```
1. ------------------------------------------------------------------
2. ---> resetting model doi to = 999.0 m depth !
3. ** Layers sliced = 7
4. ** Rho range = (66.0, 70.0, 180.0, 1000.0, 3000.0, 10000.0,
   20000.0) (Ω.m)
5. ** Minimum rho = 66.0 Ω.m
6. ** Maximum rho = 20000.0 Ω.m
7. 2021-04-17T 20:39:54 PM - Geodrill - INFO - Build the
   pseudosequences of strata.
8. ** QC flux rate = 75.0 %
9. ** Number of layers = 7
10.  ------------------------------------------------------------------
11.  Structure Rho mean value (Ω.m) Rho range (Ω.m)
12.  ------------------------------------------------------------------
13.  river zone 68.0 66.0-70.0
14.  fracture zone 123.0 66.0-180.0
15.  granite 535.0 70.0-1000.0
16.  Most Weathered 1590.0 180.0-3000.0
17.  Less Weathered 5500.0 1000.0-10000.0
18.  Igneous rocks 11500.0 3000.0-20000.0
19.  Igneous rocks 15000.0 10000.0-20000.0
20.  ------------------------------------------------------------------
21.  ** number of stations = 47
22.  ** minimum offset = 0.0 m
23.  ** maximum offset = 2300.0 m
24.  ** maximum depth = -0.0 m
25.  ** spacing depth = 39.96 m
26.  ** minumum elevation = 401.05 m
27.  ** maximum elevation = 573.4 m
28.  ** minumum resistivity value = 0.52 Ω.m
29.  ** maximum resistivity value = 49059.26 Ω.m
30.  ** Lowest station = S37
31.  ** Highest station = S00
32.  ** Altitude gap = 172.35 m
33.  ** Number of running = 2820
34.  ---> geo output file test_area.4.main._cor_oas.xlsx, has been
   successfully written to
   <C:\Users\Administrator\OneDrive\Python\pyCSAMT\_output2Oasis_>.
35.  ------------------------------------------------------------------
```

7.2.3. Export geosurface map from multi-lines from oasis outputs

With multi survey lines performed on the area, user can output multiples oasis files using the same process below in 7.2.2. From these files, user can export several depth surface maps which is a horizontal section depth map where each section depth represents the combining underground information at each line. If the depth value i.e. the depth to image is not between values of verticals z-nodes, the depth value will be interpolated. Building geosurface map can help us to checkout the existing of deep anomalies like fractures of faults and to speculated about its depth.

For instance, pour demonstration we want to export the both imaging depths (40 m and 100m , of 04 survey lines (K1 to K5)) located on 'data/inputOas' directory of the software.

Note: It's a good tip to keep or add at the end line of each oasis montaj outputs , the word "_cor_oas''. The default output format of geosurface map is 'csv' format and can be change to xlsx .

The lines of codes below allow to build a geosurface map:

```
In [88]: from pycsamt.geodrill.geoCore.geodrill import Geosurface
...: path_to_oasisfiles ='data/InputOas' # loaction of oasis
output files
...: # section depth map assumed to be 40 m and 100m .
...: output_format ='.csv'
...: values_for_imaging = [40.,100.]  # in meters
...: # we create self container of geosurface object
...: geo_surface_obj = Geosurface( path =path_to_oasisfiles,
...: depth_values = values_for_imaging,
...: )
 ...: geo_surface_obj.write_file(fileformat = output_format,
 ...: savepath =savepath )
```

When savepath is None, the default path is _outputGS_ . :

```
--------------------------GeoSurface * Data * info---------------
** ----- file : --|>K1_cor_oas :
** depth spacing = 38.0 m
** maximum depth = 912.0 m
2021-04-17T 21:08:18 PM - pycsamt.geodrill.geoCore.geodrill -
INFO - Computing profile angle from Easting and Nothing
coordinates.
** profile angle = 124.04 degrees E of N.
** geoelectric strike = 34.0 degrees E of N.
** ----- file : --|>K2_cor_oas :
** depth spacing = 38.0 m
 ** maximum depth = 912.0 m
 2021-04-17T 21:08:18 PM - pycsamt.geodrill.geoCore.geodrill -
INFO - Computing profile angle from Easting and Nothing
coordinates.
 ** profile angle = 123.86 degrees E of N.
 ** geoelectric strike = 33.0 degrees E of N.
 ** ----- file : --|>K3_cor_oas :
 ** depth spacing = 38.0 m
 ** maximum depth = 912.0 m
 2021-04-17T 21:08:18 PM - pycsamt.geodrill.geoCore.geodrill -
INFO - Computing profile angle from Easting and Nothing
coordinates.
 ** profile angle = 109.71 degrees E of N.
 ** geoelectric strike = 19.0 degrees E of N.
 ** ----- file : --|>K4_cor_oas :
 ** depth spacing = 38.0 m
```

```
22.  ** maximum depth = 912.0 m
23.  2021-04-17T 21:08:18 PM - pycsamt.geodrill.geoCore.geodrill -
   INFO - Computing profile angle from Easting and Nothing
   coordinates.
24.  ** profile angle = 109.47 degrees E of N.
25.  ** geoelectric strike = 19.0 degrees E of N.
26.  ** ----- file : --|>K5_cor_oas :
27.  ** depth spacing = 38.0 m
28.  ** maximum depth = 912.0 m
29.  2021-04-17T 21:08:18 PM - pycsamt.geodrill.geoCore.geodrill -
   INFO - Computing profile angle from Easting and Nothing
   coordinates.
30.  ** profile angle = 109.68 degrees E of N.
31.  ** geoelectric strike = 19.0 degrees E of N.
32.  ---> Geosurfaces
   outputfiles :K1K2K3K4K538.0_gs4.csv,K1K2K3K4K5114.0_gs4.csv :
   have been successfully written to
   <C:\Users\Administrator\OneDrive\Python\pyCSAMT\_outputGS_>.
```

7.3. Build Borehole or well data and/or geochemistry sample

To build borehole data from geological information and/or geochemistry sample from the survey area, user has two (02) possibilities:

- build a borehole or/and geochemistry sample log manually: Indeed, user can build stratigraphy log step by step by inputting the value of top and bottom of each stratum by responding all queries suggested by the software. When values are set, the software will generate a *well report* and build *log sheet* for other purposes. To build manually the borehole data, user must set 'auto' param from geoCore.geodrill.Drill module to False.

- build a borehole or/and geochemistry sample data automatically: User needs to provide the *parser file* to easily write borehole data. It's can be on *.csv or *. JSON format. Indeed, a parser file is a kind of file that includes the *geological layers names* with their *specific thickness*, the *borehole coordinates*, the *dip* and the *geochemistry sample*. When auto param is set to True (default), the software generated a *drillhole* data easily read by external software like *drillhole add-on module* of oasis montaj . An example of parser *.csv file (*nbleDH.csv) is* located in data/drill_example_files directory or illustration. In addition, the jocker '*' is used to build all data i.e. *drill collar*, *drill geology*, *drill geochemistry sample*, *drill elevation*,, and *azimuth*. Please refer to our documentation to get more information about Drill module.

- The following lines of codes could give the expected results:

```
In [91]: # if set to False , user will add step by step all data
with the layer thicknesses
...: build_borehole_auto=True
...: # create a borehole object
...: borehole_obj = Drill (well_filename=
'data/drill_example_files/nbleDH.csv',
...: auto= build_borehole_auto)
...: # data2write : which kind of data do you want to output ?
...: # borehole collar, geology? borehole geochemistry sample ?
or borehole survey elevation ?
...: kind_of_data2output = '*' # can be 'collar' `geology` ,
`sample`
...: borehole_obj.writeDHData(data2write=kind_of_data2output,
 ...: savepath = savepath )
```

The default savepath is '_ouputDH_

```
2021-04-17T 21:45:27 PM - pycsamt.utils.func_utils - INFO - You
pass by _order_well function! Thin now , everything is ok.
2021-04-17T 21:45:27 PM - pycsamt.utils.func_utils - INFO - You
pass by _order_well function! Thin now , everything is ok.
---> Borehole output <nbledhxlsx> has been written to
C:\Users\Administrator\OneDrive\Python\pyCSAMT\_outputDH_.
```

7.4. Case study: combined new model resistivity and geosurface map

The profile map (figure 14a) as well as the stacked models sections maps (Figures 5b-6b) emphasize the different structural dips, and to apprehend the nature of the underlying layers located in the area. The geosurface map (D-494) at 494 meters deep (about half of the depth of investigation) visible on stacked sections, helps to understand the deeper of existing fractures and faults in the area. Overall, figure 14 shows the 05 CSAMT survey lines. These lines are substantially paralleling about 126 ° NW direction and secant to the main fault F1. Figure 14b displays the presence of conductive anomalies around stations S06, S12, S27, S42, and stations S04, S10 S15 on line 01 and line 02 respectively. We finally could deduce that these observed anomalies, are local abnormal body responses with low resistivity values. Moreover, the presence of numerous fragmented rocks and the existence of micro-fractures (see figure 09 of pseudo-cross-section of) along different station of line 01, are assumed to be the results of metamorphism accelerating process.

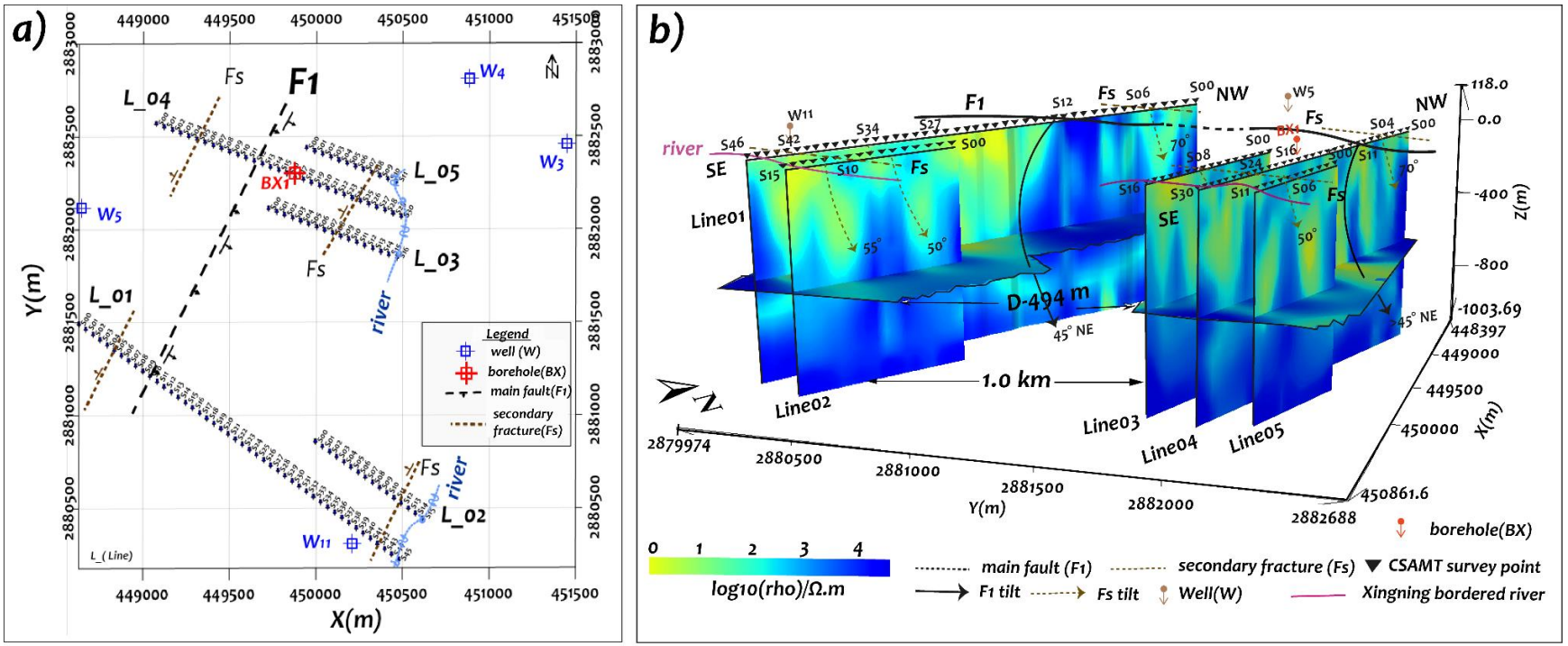

Figure 14) a) Geosurface maps; line 01 and line 02 oriented 126 ° while line 03 to line 05 oriented 110°, b) Stacked sections map from CSAMT 2D inversion. Dip angle of main fault (F1) is represented in perspective view.

8. GeoDB : Update and upgrade the geodatabase

geoDB or *geodatabase* integrates geological data obtained from field, wells and/or boreholes data into its own SQL database. *The sub-package* considers different geological *layer names*, *station locations*, and different 2D inversions results with their corresponding parameters. Each geological formation (rock or stratum) and its own *electrical resistivity properties*, its *digital cartography map* symbolization called FGDC (Digital cartographic Standard for Geological Map Symbolization), its *pattern*, its *geological code*, as well as its *label*, are included into the database. To update and updgrade the geodatabase , please refer to our wiki pages : https://github.com/WEgeophysics/pyCSAMT/wiki/How-pyCSAMT-works-%3F#update-and-upgrade-the-geodatabase

Conclusion

We just give an overview of main functionalities of pyCSAMT, however, browsing the package will let user to discover others add-on modules and their uses. Have fun!