

# MTPy User Guide

October 10, 2018

## 1 Introduction

This workbook contains some examples for reading, analysing and plotting processed MT data. It covers most of the steps available in MTPy. For more details on specific input parameters and other functionality, we recommend looking at the mtpy documentation, which can be found at: [https://mtpy2.readthedocs.io/en/develop/master\\_doc.html](https://mtpy2.readthedocs.io/en/develop/master_doc.html).

This workbook is structured according to some of the key modules in MTPy: Core, Analysis, Imaging, and Modeling.

### 1.1 Getting Started

To start with, you will need to make sure MTPy is installed and is working correctly. Please see the installation guide (<https://github.com/MTgeophysics/mtpy/wiki/MTPy-installation-guide-on-Windows-PC>) for details.

Before you begin these examples, we suggest you make a temporary folder (e.g. C:/tmp) to save all example outputs.

### 1.2 Useful tricks and tips

This workbook exists as a Jupyter notebook and a pdf. If you are running the Jupyter notebook, you can run each of the cells, modifying the inputs to suit your requirements. Most of these examples have been written to be self contained.

In Jupyter, you can add the following line to the top of any cell and it will write the contents of that cell to a python script: `%%writefile example.py`

Many of the examples below make use of the matplotlib colour maps. Please see [https://matplotlib.org/examples/color/colormaps\\_reference.html](https://matplotlib.org/examples/color/colormaps_reference.html) for colour map options.

## 2 Core

These first few examples cover some of the basic functions and tools that can be used to look at data contained in an edi file, plot it, and make changes (e.g. sample onto different frequencies).

### 2.1 Read an edi file into an MT object

```
In [1]: # import required modules
        from mtpy.core.mt import MT
```

```
# Define the path to your edi file
edi_file = "C:/mtpywin/mtpy/examples/data/edi_files_2/Synth00.edi"

# Create an MT object
mt_obj = MT(edi_file)
```

The mt\_obj contains all the data from the edi file, e.g. impedance, tipper, frequency as well as station information (lat/long). To look at any of these parameters you can type, for example:

```
In [2]: # To see the latitude and longitude
        print mt_obj.lat, mt_obj.lon
```

```
-19.01 136.01
```

```
In [3]: # To see the easting, northing, and elevation
        print mt_obj.east, mt_obj.north, mt_obj.elev
```

```
606300.40602 7897760.86059 95.0
```

There are many other parameters you can look at in the mt\_obj. Just type mt\_obj.[TAB] to see what is available. In the MT object are the Z and Tipper objects (mt\_obj.Z; mt\_obj.Tipper). These contain all information related to, respectively, the impedance tensor and the tipper.

```
In [4]: # for example, to see the frequency values represented in the impedance tensor:
        print mt_obj.Z.freq
```

```
[1.256500e+04 9.751601e+03 7.876300e+03 6.188500e+03 5.250801e+03
 4.265799e+03 3.515799e+03 8.437800e+02 6.562798e+02 4.922399e+02
 3.867599e+02 3.164400e+02 2.578400e+02 2.109600e+02 1.728900e+02
 1.367200e+02 1.015600e+02 7.421900e+01 5.761700e+01 4.882800e+01
 4.101600e+01 3.222700e+01 2.636700e+01 2.148400e+01 1.757800e+01
 1.440400e+01 1.147500e+01 8.593800e+00 6.591801e+00 5.371100e+00
 4.394500e+00 3.601100e+00 2.868700e+00 2.304700e+00 1.914100e+00
 1.601600e+00 1.328100e+00 1.074200e+00 8.789100e-01 6.835900e-01
 5.078100e-01 3.710900e-01 2.880900e-01 2.050800e-01 1.318400e-01
 8.789098e-02 6.835900e-02 5.127000e-02 4.028299e-02 3.295900e-02
 2.685500e-02 2.197300e-02 1.709000e-02 1.281700e-02 1.007100e-02
 8.239700e-03 6.713900e-03 5.493201e-03 4.272499e-03 2.822900e-03
 2.059900e-03 1.678500e-03 1.373300e-03 1.068100e-03 7.629400e-04]
```

```
In [5]: # or to see the impedance tensor (first 4 elements)
        print mt_obj.Z.z[:4]
```

```
[[[ 26.58566 -4.302123j 482.4492 +604.7747j ]
 [-410.0502 -800.4257j 8.994784 +44.07396j ]]
```

```
[[ 12.43271 +7.519158j 434.8246 +514.6176j ]
 [-372.7205 -666.402j   17.64062 +36.09528j ]]

[[ 7.652151 +6.28703j 398.3996 +460.0998j ]
 [-349.9875 -580.3959j 21.57495 +33.98854j ]]

[[ 3.59474 +1.225811j 362.5121 +413.2823j ]
 [-328.0029 -501.5329j 25.02421 +33.02813j ]]]
```

```
In [6]: # or the resistivity or phase (first 4 values)
```

```
print mt_obj.Z.resistivity[:4]
print mt_obj.Z.phase[:4]
```

```
[[[1.15448560e-02 9.52661629e+00]
 [1.28742136e+01 3.22072438e-02]]
```

```
[4.32975088e-03 9.30931663e+00]
 [1.19572611e+01 3.31035019e-02]]
```

```
[2.49056438e-03 9.40578869e+00]
 [1.16641228e+01 4.11538240e-02]]
```

```
[4.66179794e-04 9.76706091e+00]
 [1.16060807e+01 5.54922342e-02]]]
[[[-9.19198953 51.41945343]
 [-117.1256163 78.46525668]]
```

```
[31.16505948 49.80396162]
 [-119.21840697 63.95414249]]
```

```
[39.40662667 49.11076951]
 [-121.09061209 57.59379146]]
```

```
[18.82944281 48.74426019]
 [-123.18471678 52.85011776]]]
```

As with the MT object, you can explore the object by typing `mt_obj.Z.[TAB]` to see the available attributes.

## 2.2 Plot an edi file

In this example we plot MT data from an edi file.

```
In [7]: # import required modules
from mtpy.core.mt import MT
import os
```

```

# Define the path to your edi file and save path
edi_file = "C:/mtpywin/mtpy/examples/data/edi_files_2/Synth00.edi"
savepath = r"C:/tmp"

# Create an MT object
mt_obj = MT(edi_file)

# To plot the edi file we read in in Part 1 & save to file:
pt_obj = mt_obj.plot_mt_response(plot_num=1, # 1 = yx and xy; 2 = all 4 components
                                  # 3 = off diagonal + determinant
                                  plot_tipper = 'yri',
                                  plot_pt = 'y' # plot phase tensor 'y' or 'n'
                                  )
#pt_obj.save_plot(os.path.join(savepath, "Synth00.png"), fig_dpi=400)

```

<Figure size 960x720 with 5 Axes>

## 2.3 Make some change to the data and save to a new file

This example demonstrates how to resample the data onto new frequency values and write to a new edi file. In the example below, you can either choose every second frequency or resample onto five periods per decade. To do this we need to make a new Z object, and save it to a file.

```

In [8]: # import required modules
from mtpy.core.mt import MT
import os

# Define the path to your edi file and save path
edi_file = r"C:/mtpywin/mtpy/examples/data/edi_files_2/Synth00.edi"
savepath = r"C:/tmp"

# Create an MT object
mt_obj = MT(edi_file)

# First, define a frequency array:
# Every second frequency:
new_freq_list = mt_obj.Z.freq[:,2]

# OR 5 periods per decade from 10^-4 to 10^3 seconds
from mtpy.utils.calculator import get_period_list
new_freq_list = 1./get_period_list(1e-4,1e3,5)

# Create new Z and Tipper objects containing interpolated data
new_Z_obj, new_Tipper_obj = mt_obj.interpolate(new_freq_list)

# Write a new edi file using the new data
mt_obj.write_mt_file(

```

```

save_dir=savepath,
fn_basename='Synth00_5ppd',
file_type='edi',
new_Z_obj=new_Z_obj, # provide a z object to update the data
new_Tipper_obj=new_Tipper_obj, # provide a tipper object
longitude_format='LONG', # write longitudes as 'LONG' not 'LON'
latlon_format='dd' # write as decimal degrees (any other input
                    # will write as degrees:minutes:seconds
)

```

```

C:\mtpywin\mtpy\mtpy\utils\calculator.py:322: RuntimeWarning: invalid value encountered in double
z_rel_err = error/z_amp

```

```

Out[8]: 'C:\\tmp\\Synth00_5ppd_1.edi'

```

### 3 Analysis

The analysis modules contain functions to analyse MT data, for example analyse dimensionality or strike angle for either a single edi file or a collection of edi files.

#### 3.1 Look at the Phase Tensor

The phase tensor is an object within the MT object. So when you read in an edi file, it will automatically contain the phase tensor. To look at the attributes of the phase tensor, you can (as before) type `mt_obj.[TAB]` to see what is available. For example, to look at the skew angle (beta):

```

In [1]: from mtpy.core.mt import MT
import matplotlib.pyplot as plt

# Define the path to your edi file and save path
edi_file = r"C:/mtpywin/mtpy/examples/data/edi_files_2/Synth00.edi"

mt_obj = MT(edi_file)

# look at the skew values as a histogram
plt.hist(mt_obj.pt.beta,bins=50)
plt.xlabel('Skew angle (degree)')
plt.ylabel('Number of values')

```

```

Out[1]: Text(0,0.5,'Number of values')

```

#### 3.2 Look at dimensionality

Here, we use the phase tensor to determine which parts of the impedance tensor are 1D, 2D or 3D. The code returns an array with a value (1, 2 or 3) representing the dimensionality

[illegible]

### 3.3 Look at strike angle

Here, we calculate the strike angle for the 2D part of the impedance tensor.

```
In [3]: # Import required modules
        from mtpy.analysis.geometry import strike_angle
        from mtpy.core.mt import MT
        import numpy as np
        import os

        # Define the path to your edi file and save path
        edi_file = r"C:/mtpywin/mtpy/examples/data/edi_files_2/Synth00.edi"
        savepath = r"C:/tmp"

        # Create an MT object
        mt_obj = MT(edi_file)

        # calculate strike
        strike = strike_angle(z_object = mt_obj.Z,
                              skew_threshold = 5, # threshold for 3D
                              eccentricity_threshold=0.1 # threshold for 2D
                              )

        # display the median strike angle for this station
        # two values because of 90 degree ambiguity in strike
        print np.nanmedian(strike,axis=0)
```

```
[-4.50130871 85.49869129]
```

### 3.4 Use dimensionality to mask an edi file

Here, we use the dimensionality calculated in Part 1 to mask all 3D parts of the impedance tensor and write to a new edi file.

```
In [4]: # Import required modules
from mtpy.core.z import Z, Tipper
from mtpy.analysis.geometry import dimensionality
from mtpy.core.mt import MT
import os

# Define the path to your edi file and save path
edi_file = r"C:/mtpywin/mtpy/examples/data/edi_files_2/Synth00.edi"
savepath = r"C:/tmp"

# Create an MT object
mt_obj = MT(edi_file)

# Create an array (mask) with value True where the data are 1D or 2D and
# False where the data are 3D
dim = dimensionality(z_object = mt_obj.Z,
                    skew_threshold = 5,
                    eccentricity_threshold=0.1)

mask = dim < 3

# Apply masking. The new arrays z_array, z_err_array, and freq will
# exclude values where mask is False (i.e. the 3D parts)
new_Z_obj = Z(z_array=mt_obj.Z.z[mask],
             z_err_array=mt_obj.Z.z_err[mask],
             freq=mt_obj.Z.freq[mask])
new_Tipper_obj = Tipper(tipper_array=mt_obj.Tipper.tipper[mask],
                      tipper_err_array=mt_obj.Tipper.tipper_err[mask],
                      freq = mt_obj.Tipper.freq[mask])

# Write a new edi file (as before)
mt_obj.write_mt_file(save_dir=savepath,
                    fn_basename='Synth00_mask3d',
                    file_type='edi',
                    new_Z_obj=new_Z_obj,
                    new_Tipper_obj=new_Tipper_obj,
                    longitude_format='LONG',
                    latlon_format='dd')
```

```
Out[4]: 'C:\\tmp\\Synth00_mask3d_2.edi'
```

### 3.5 Plot strike for a collection of files

In this example, we plot the strike as a rose plot for a collection of edi files. Strike is calculated from the Z invariants (after Weaver et al., 2000; 2003), the phase tensor (Caldwell et al. 2004) and the tipper. Plots can include all frequencies or be separated according to decade if you set `plot_type = 1`.

```
In [5]: # Import required modules
        from mtpy.imaging.plotstrike import PlotStrike
        import os

        # Path containing edi files
        edi_path = r'C:/mtpywin/mtpy/examples/data/edi_files_2'

        # Full path to save path if you want to save the image
        savepath = r'C:/tmp'

        # Get full path to all files with the extension '.edi' in edi_path
        edi_list = [os.path.join(edi_path,ff) for ff in os.listdir(edi_path) \
                    if ff.endswith('.edi')]

        # make a plot (try also plot_type = 1 to plot by decade)
        strikeplot = PlotStrike(fn_list=edi_list,
                               plot_type=2,
                               plot_tipper='y')

        # save to file
        strikeplot.save_plot(savepath,
                             file_format='.png',
                             fig_dpi=400)
```

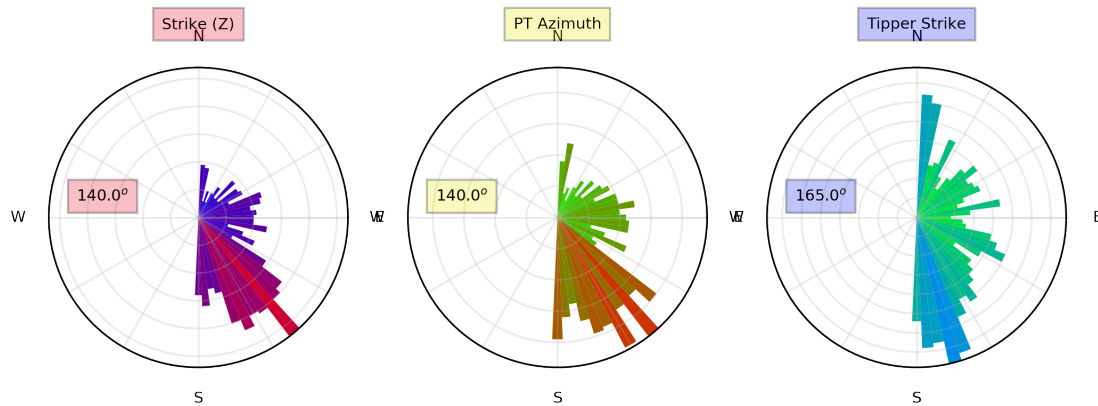
Reading 28 stations

```
C:\mtpywin\mtpy\mtpy\analysis\zinvariants.py:226: RuntimeWarning: invalid value encountered in a
strikeangerr = abs(.5 * np.arcsin(inv7)) * (180 / np.pi)
C:\mtpywin\mtpy\mtpy\analysis\pt.py:575: RuntimeWarning: invalid value encountered in sqrt
az_err = np.sqrt(self.alpha+self.beta)
```

```
-----Period Range 1e-05 to 1e+03 (s)-----
*Z-Invariants:  median=125.2 mode=140.0 mean=140.0
*PT Strike:      median=125.8 mode=140.0 mean=109.6
*Tipper Strike:  median=109.5 mode=165.0 mean=99.9
```

Note: North is assumed to be 0 and the strike angle is measured clockwise positive.





Saved figure to: C:/tmp\StrikeAnalysis.png

<Figure size 432x288 with 0 Axes>

### 3.6 Rotating an edi file to strike and save to a new file

In the example below, we read an edi file, rotate it to a pre-defined strike and save it to a new file. Before saving the new edi file, we display the new rotation angle. This step can be omitted, but it is useful to check that the final rotation angle for the edi file is what you expect. Note there are multiple values for rotation angle representing each frequency contained in the edi file.

```
In [6]: # Import required modules
from mtpy.core.z import Z, Tipper
from mtpy.analysis.geometry import strike_angle
from mtpy.core.mt import MT
import os

# Define the path to your edi file and save path
edi_file = r"C:/mtpywin/mtpy/examples/data/edi_files_2/Synth00.edi"
savepath = r"C:/tmp"

# Create an MT object
mt_obj = MT(edi_file)

# From 3.3; we determine strike angle is -4 or 86 degrees
# based on other geological information we determine it to be
# -4 degrees.
strike = -4
mt_obj.Z.rotate(strike)
mt_obj.Tipper.rotate(strike)
```

```

# check the rotation angle
print mt_obj.Z.rotation_angle

# Write a new edi file (as before)
mt_obj.write_mt_file(save_dir=savepath,
                    fn_basename='Synth00_rotate%1i'%strike,
                    file_type='edi',
                    longitude_format='LONG',
                    latlon_format='dd')

[356. 356. 356. 356. 356. 356. 356. 356. 356. 356. 356. 356. 356.
 356. 356. 356. 356. 356. 356. 356. 356. 356. 356. 356. 356.
 356. 356. 356. 356. 356. 356. 356. 356. 356. 356. 356. 356.
 356. 356. 356. 356. 356. 356. 356. 356. 356. 356. 356. 356.
 356. 356. 356. 356. 356. 356. 356. 356.]

```

```
Out[6]: 'C:\\tmp\\Synth00_rotate-4_1.edi'
```

## 4 Imaging

### 4.1 Plotting penetration depth using the Niblett-Bostick transform

#### 4.1.1 Penetration depth as a function of period for 1 site

Here, we plot the Niblett-Bostick depth as a function of period/frequency for one site.

```

In [1]: # Import required modules
import os
from mtpy.imaging import penetration_depth1d as pd1d

# Define edi file
edi_file = r"C:/mtpywin/mtpy/examples/data/edi_files_2/Synth00.edi"

# Define file to save to
save_file = r"C:/tmp/penetration_depth1d.png"

# Create a plot of penetration depth and save to file
pd1d.plot_edi_file(edi_file, savefile=save_file, fig_dpi=400)

```

<Figure size 640x480 with 1 Axes>

#### 4.1.2 Penetration depth for selected periods along a profile

Here, we plot the Niblett-Bostick depth as a function of distance along a profile, for selected periods

```

In [2]: #Import required modules
        from mtpy.imaging import penetration_depth2d as pen2d

        #Define a path containing all the edi files in the profile & a savepath
        edi_path = r'C:/mtpywin/mtpy/data/edifiles'
        savepath = r'C:/tmp'

        # Choose indices of periods to plot
        period_index_list = [0, 1, 10, 20, 30, 40, 50, 59]

        # Plot profiles for different modes ('det', 'zxy' or 'zyx')
        pen2d.plot2Dprofile(edi_path, period_index_list,
                           'det', marker='o',
                           tick_params={'rotation':'vertical'})

        # To save any of these figures:
        plt.savefig(os.path.join(savepath, 'penetration_depth_profile.png'), dpi=400)

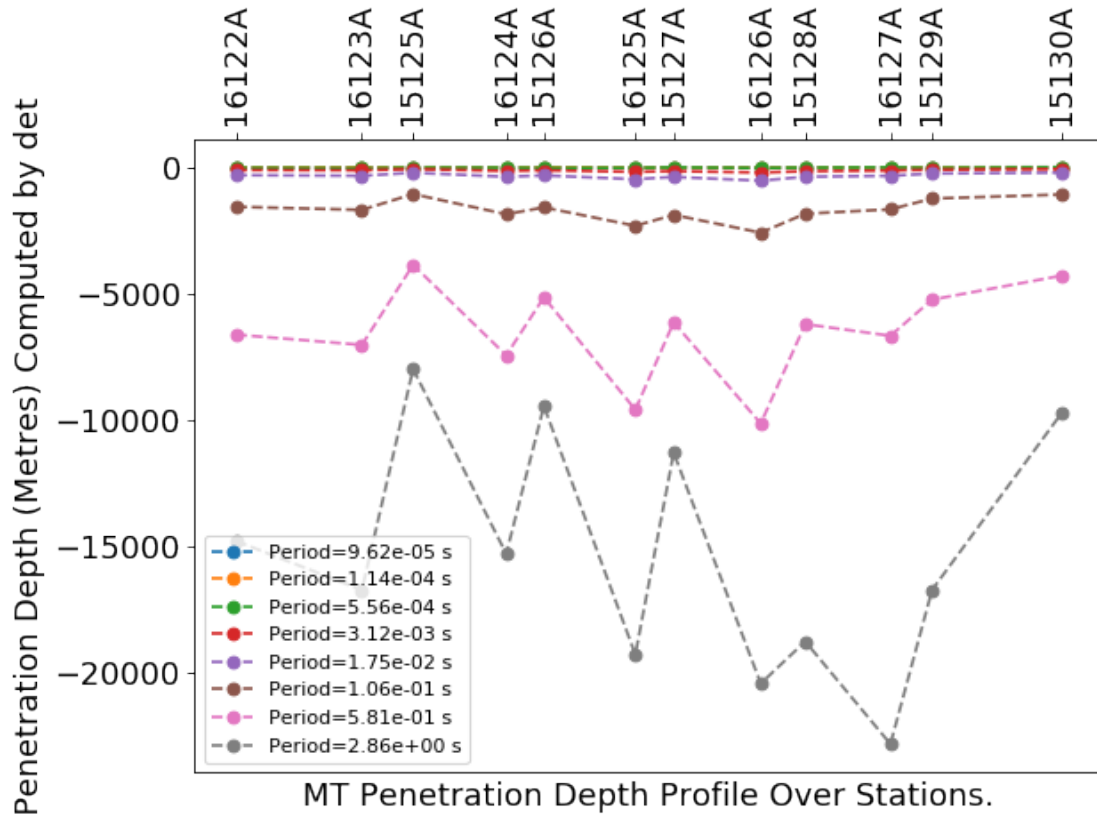
```

Should input a freq array to know which index of the PT array corresponds to which freq.  
Should input a freq array to know which index of the PT array corresponds to which freq.  
Should input a freq array to know which index of the PT array corresponds to which freq.  
Should input a freq array to know which index of the PT array corresponds to which freq.  
Should input a freq array to know which index of the PT array corresponds to which freq.  
Should input a freq array to know which index of the PT array corresponds to which freq.  
Should input a freq array to know which index of the PT array corresponds to which freq.  
Should input a freq array to know which index of the PT array corresponds to which freq.  
Should input a freq array to know which index of the PT array corresponds to which freq.  
Should input a freq array to know which index of the PT array corresponds to which freq.  
Should input a freq array to know which index of the PT array corresponds to which freq.

=====

Rotated Z and Tipper to align with +171.87 degrees E of N  
Profile angle is +261.87 degrees E of N  
=====

C:\mtpywin\Anaconda2\lib\site-packages\matplotlib\figure.py:2299: UserWarning: This figure includes Axes that are not compatible "



#### 4.1.3 Penetration depth for a selected period as a map

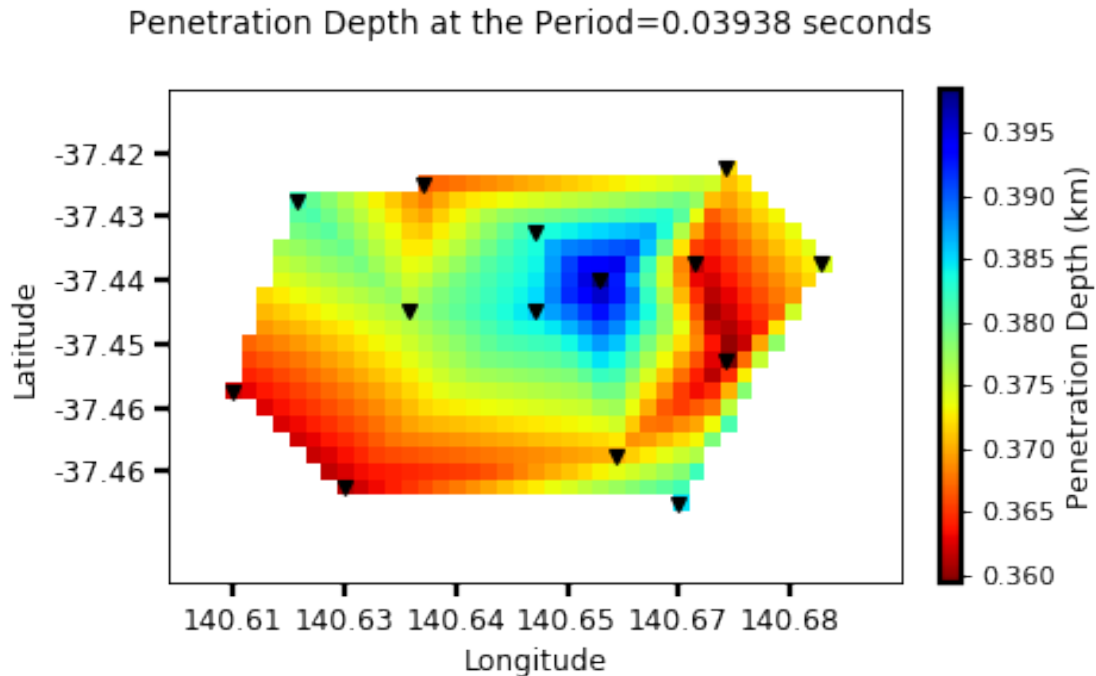
Here, we plot the Niblett-Bostick depth for one period, and interpolate between stations on a map.

```
In [3]: #Import required modules
from mtpy.imaging import penetration_depth3d as pen3d

# Set path to edi files to include in the plot and define savepath
edi_path = r'C:/mtpywin/mtpy/examples/data/edi2'
savepath = r'C:/tmp'

# Create plot for period index number 10 for determinant
pen3d.plot_latlon_depth_profile(edi_path, 10, 'det', showfig=True,
                                savefig=True, savepath=savepath,
                                fontsize=11, fig_dpi=400, file_format='png')
```

```
C:\mtpywin\mtpy\mtpy\imaging\penetration.py:369: RuntimeWarning: invalid value encountered in le
grid_z[grid_z < 0] = np.nan
```



C:\mtpywin\mtpy\mtpy\imaging\penetration.py:474: DeprecationWarning: Call to deprecated function  
 def get\_period\_fmt(self):

## 4.2 Phase tensor plotting

In the next examples we will plot phase tensor maps (after Caldwell et al., 2004 & Bibby et al. 2005) and induction vectors. In these and later maps, there are many different parameters that can be varied, and generally there will be some tweaking that will need to be done to get the parameters (e.g. ellipse scaling) perfect for publication-quality plots. Note that some parameter tweaks can be done later (e.g. in Adobe Illustrator or Inkscape) if you save the plot to a vector graphic format like .eps and we recommend this as it can take hours to get all the parameters perfect!

### 4.2.1 Phase tensor maps

In this example we plot all edi files from a directory as a phase tensor map for a selected frequency, with real induction vectors (Parkinson convention) overlain. There are quite a few parameters you can adjust, feel free to spend some time with this example updating them and viewing the result.

```
In [4]: # Import required modules
        from mtpy.imaging.phase_tensor_maps import PlotPhaseTensorMaps
        import os

        import matplotlib.pyplot as plt
```

```

# directory containing edis
edi_path = r'C:/mtpywin/mtpy/examples/data/edi2'

# full path to file to save to
savepath = r'C:/tmp'

# frequency to plot
plot_freq = 1e-2

# name image according to plot frequency
image_fn = 'phase_tensor_map%1is'%(int(1./plot_freq))+'.png'

# gets edi file names as a list
edi_list = [os.path.join(edi_path,ff) for ff in os.listdir(edi_path) \
            if ff.endswith('.edi')]

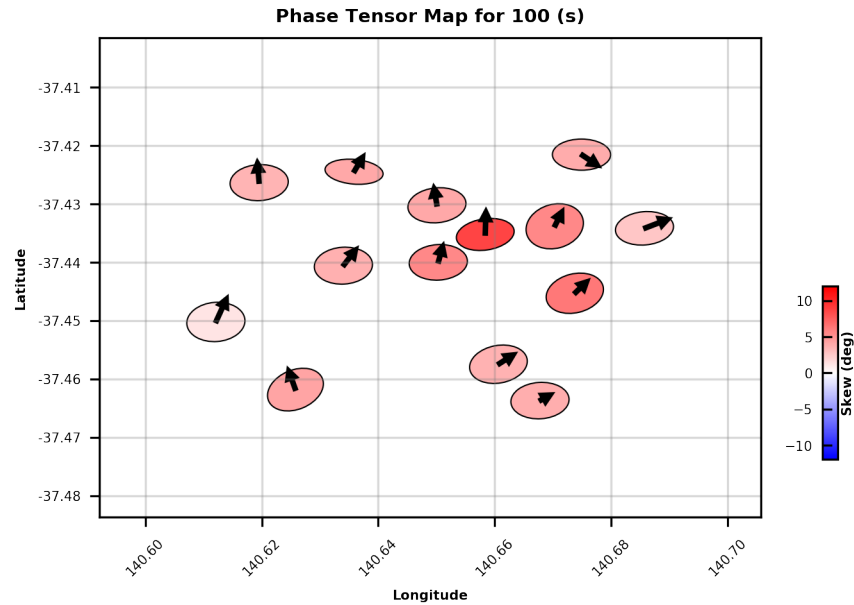
# generate plot
ptmap = PlotPhaseTensorMaps(fn_list = edi_list,
                             plot_freq = plot_freq , # frequency to plot
                             fig_size = (4,3), # x, y dimensions of figure
                             xpad = 0.02, ypad = 0.02, # pad around stations
                             plot_tipper = 'yr', # 'y' + 'r' and/or 'i' to plot
                                                # real and/or imaginary
                             edgecolor='k', # a matplotlib colour or None for no borders
                             lw=0.5, # linewidth for the ellipses
                             minorticks_on=False, # whether or not to turn on minor ticks
                             ellipse_colorby='skew', # 'phimin', 'phimax', or 'skew'
                             ellipse_range = [-12,12,2], # [min,max,step]
                             ellipse_size=0.01, # scaling factor for the ellipses
                             arrow_size=0.1,
                             arrow_head_width=0.002, # scaling for arrows (head width)
                             arrow_head_length=0.002, # scaling for arrows (head length)
                             ellipse_cmap='bwr', # matplotlib colormap
                             station_dict={'id':(5,7)} ,
                             cb_dict = {'position':
                                         [1.05,0.2,0.02,0.3]}, # colorbar position [x,y,dx,dy]
                             font_size=5
                             )

# save the plot
ptmap.save_figure(os.path.join(savepath,image_fn))

```

Reading 14 stations

C:\mtpywin\mtpy\mtpy\utils\calculator.py:322: RuntimeWarning: invalid value encountered in double  
 $z\_rel\_err = error/z\_amp$



<Figure size 432x288 with 0 Axes>

If you want to plot some other image in the background (e.g. gravity) you can load it as follows & plot (the example below uses a Bouguer anomaly grid in netCDF format)

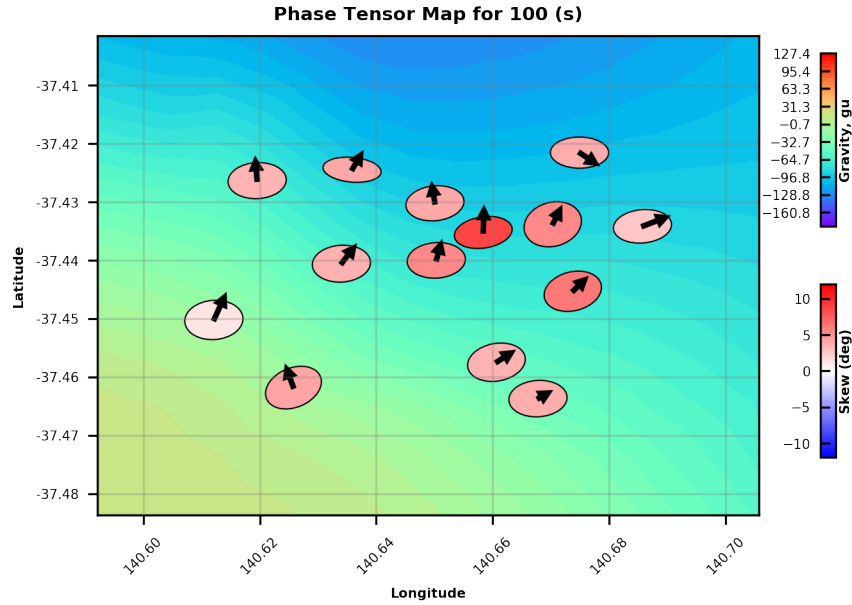
**Note** For this example you will need to install netcdf4 if it's not there already. Type the following into your command prompt:

conda install netcdf4

```
In [5]: # Import the netCDF4 module
from netCDF4 import Dataset
import numpy as np

# load a netcdf file
grav = Dataset(r'C:/mtpywin/mtpy/examples/data/gravity/gravity.nc')
glon, glat = np.meshgrid(grav.variables['lon'], grav.variables['lat'])
vals = grav.variables['Band1'][:]
```

```
ptmap.plot(show=True,
           raster_dict={'lons': glon.flatten(),
                        'lats': glat.flatten(),
                        'vals': vals.flatten(),
                        'levels': 50,
                        'cmap': 'rainbow',
                        'cbar_title': 'Gravity, gu',
                        'cbar_position': [1.05, 0.6, 0.02, 0.3]})
```



#### 4.2.2 Phase tensor sections

In this example we plot phase tensors for all frequencies from some edi files located along a transect.

```
In [6]: # import required modules
from mtpy.imaging.phase_tensor_pseudosection import PlotPhaseTensorPseudoSection
import os

# path to edis
edi_path = r'C:/mtpywin/mtpy/examples/data/edi_files_2'

# save path
savepath = r'C:/tmp'

# edi list
edi_list=[os.path.join(edi_path,edi) for edi in os.listdir(edi_path) \
          if ((edi.endswith('.edi')))]

# dictionary containing ellipse properties
ellipse_dict = {'ellipse_colorby':'phimin', # phimin, phimax, skew,
               # skew_seg
               'ellipse_range':[0,90]} # Color limits. If plotting
                                     # skew_seg need to provide
                                     # 3 numbers, the 3rd indicates
                                     # interval, e.g. [-12,12,3]
```



```

# create a plot object
ptsection = PlotPhaseTensorPseudoSection(fn_list = edi_list,
    linedir='ns', # 'ns' if the line is closer to north-south,
                # 'ew' if line is closer to east-west
    stretch=(17,8), # determines (x,y) aspect ratio of plot
    station_id=(0,10), # indices for showing station names
    plot_tipper = 'yri', # plot tipper ('y') + 'ri' for real+imag
    font_size=5,
    lw=0.5,

)

ptsection.ellipse_size = 2.
ptsection.plot()

ptsection.save_figure(save_fn = os.path.join(savepath, 'PhaseTensorSection.pdf'),
    fig_dpi=400)

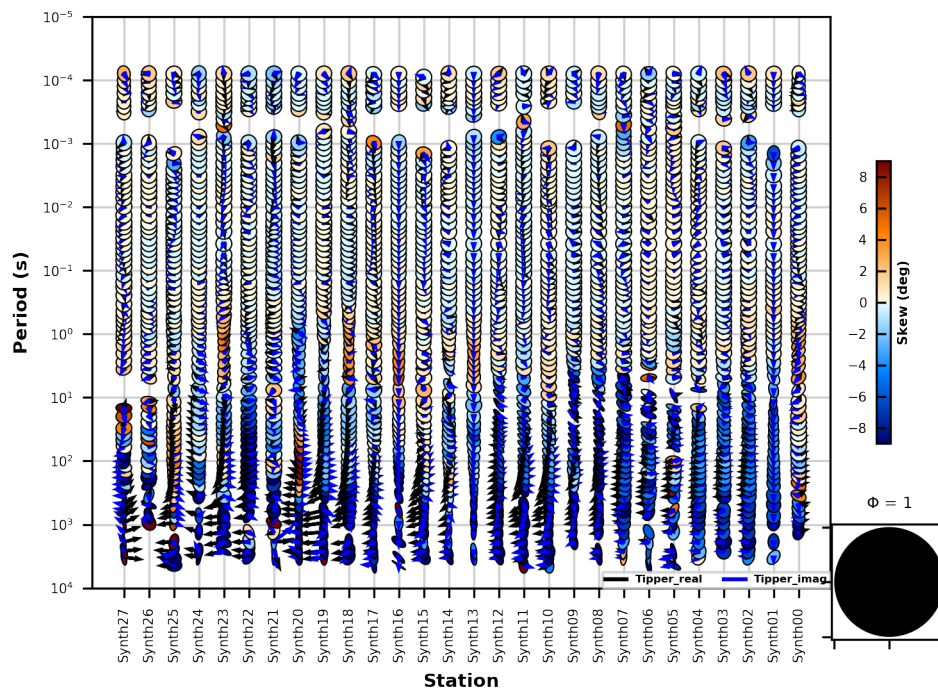
```

Reading 28 stations

```

-----
skew min = -76.25
skew max = 76.64
-----

```



Saved figure to: C:/tmp/PhaseTensorSection.pdf

<Figure size 432x288 with 0 Axes>

## 4.3 Apparent resistivity and phase sections and maps

### 4.3.1 Apparent resistivity maps

Plot apparent resistivity for a selected period, for an array of sites. Resistivity values are interpolated between sites

```
In [7]: # Import required modules
        from mtpy.imaging.plot_resphase_maps import PlotResPhaseMaps
        import os

        # Define edi path and save path
        edipath = r'C:/mtpywin/mtpy/examples/data/edi2'
        savepath = r'C:/tmp'

        # Make an edi list
        edi_list = [os.path.join(edipath,ff) for ff in os.listdir(edipath) \
                    if ff.endswith('.edi')]

        # Define plot frequency
        freq = 0.001

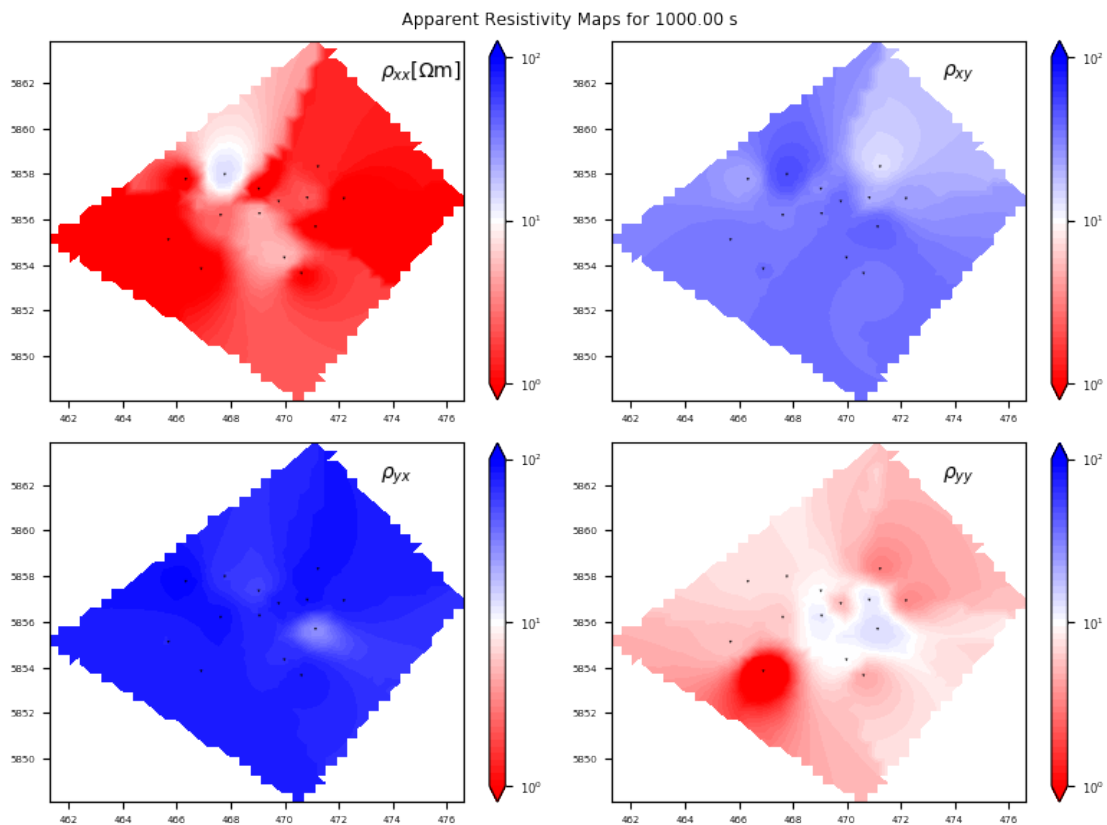
        # Initialise a plot object
        prp = PlotResPhaseMaps(fn_list=edi_list,
                               mapscale='km')

        # plot resistivity
        f = prp.plot(freq,
                     'res',
                     1, 100, # limits in resistivity. Can be provided as a number
                              # or a 2x2 array to give different limits for
                              # different components
                     nn=3, p=3,
                     extrapolation_buffer_degrees=0.05, # how far to extrapolate
                     regular_grid_nx=40, # number of x grid points
                     regular_grid_ny=40, # number of y grid points
                     cmap='bwr_r', # colour map
                     show=True)

        # save plot
        prp.fig.savefig(os.path.join(savepath, 'Resistivity_%1s.png'%(1./freq)),
                        dpi=300)
```

Reading 14 stations

C:\mtpywin\Anaconda2\lib\site-packages\matplotlib\contour.py:1004: UserWarning: The following kw  
s)

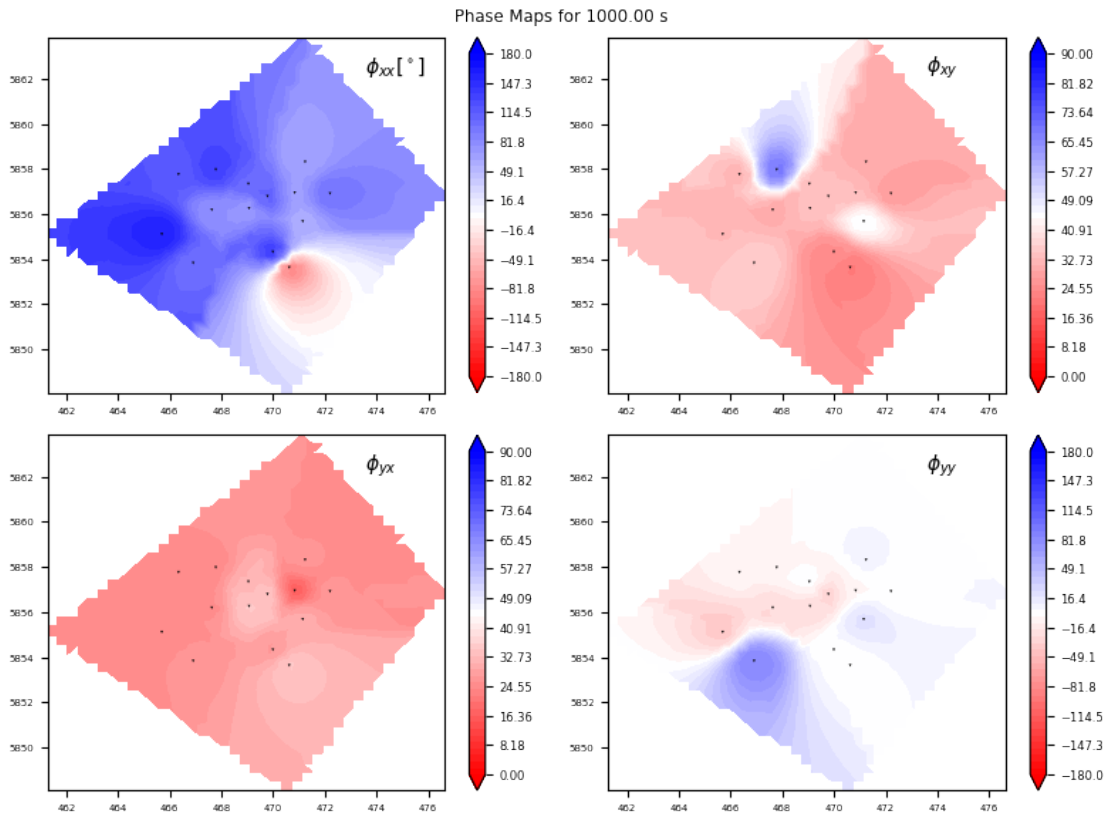


### 4.3.2 Phase maps

Similarly, you can plot phase as an interpolated map.

```
In [8]: # plot phase
f = prp.plot(freq, # frequency to plot
             'phase', # phase or res
             np.array([[-180,0], # minimum phase
                       [0,-180]]), # for colour stretch
             np.array([[180,90], # maximum phase
                       [90,180]]), # for colour stretch
             nn=3, p=3,
             extrapolation_buffer_degrees=0.05, # how far to extrapolate
             regular_grid_nx=40, # number of x grid points
             regular_grid_ny=40, # number of y grid points
             cmap='bwr_r', # colour map
             show=True)
```

```
# save plot
prp.fig.savefig(os.path.join(savepath, 'Phase_%1is.png'%(1./freq)),dpi=300)
```



## 4.4 Apparent resistivity and phase pseudosections

Plot apparent resistivity and phase as a function of period for several sites located along a transect

```
In [9]: # Import required modules
import os
import mtpy.imaging.plotpseudosection as pps

# define edi path and savepath
edi_path = r'C:/mtpywin/mtpy/examples/data/edi_files'
savepath = r'C:/tmp'

# list all edi files in directory
edi_list = [os.path.join(edi_path,ff) for ff in os.listdir(edi_path) \
            if (ff.endswith('.edi'))]

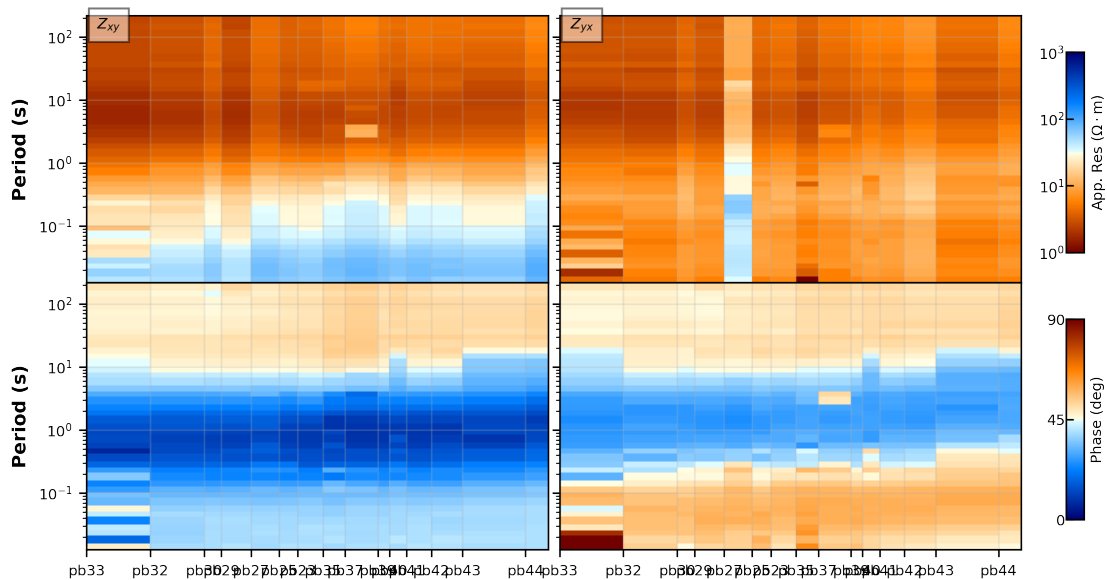
# make a plot
```

```

pps.PlotResPhasePseudoSection(fn_list = edi_list,
                              linedir='ns', # 'ns' or 'ew' - approximate line
                                           # orientation
                              plot_xx = 'n', # plot xx component 'y' or 'n'
                              plot_xy = 'y', # plot xy component 'y' or 'n'
                              plot_yx = 'y', # plot yx component 'y' or 'n'
                              plot_yy = 'n', # plot yy component 'y' or 'n'
                              res_limits=[0,3], # log resistivity limits
                              phase_limits=[0,90], # log phase limits
                              shift_yx_phase = True, # True or False
                              plot_style='pcolormesh' # 'pcolormesh' or 'imshow'
                              )

```

Reading 15 stations



Out[9]: <mtpy.imaging.plotpseudosection.PlotResPhasePseudoSection at 0x12bb4390>

## 5 Modeling

The modeling modules in MTPy are designed to support third party software. I.e. they assist with input file generation and then analysis and visualisation of the results. Here, we give examples the three more commonly used software packages. Occam 1d, Occam 2d and ModEM.

### 5.1 Occam 1D

This first example demonstrates how to create input files for inversion of MT data using the Occam 1D code (Key 2009, Constable et al 1987) - <http://marineemlab.ucsd.edu/Projects/Occam/1DCSEM/>.

There are three input files. \* Data file, containing the data for inversion \* Model file (which contains the model mesh) \* Startup file - called by Occam1d and contains the data and model file names together with some control parameters for the inversion.

```
In [1]: # import required modules
import os
import mtpy.modeling.occam1d as mtoc1d

# full path to edi file and save path
edi_file = r'C:/mtpywin/mtpy/examples/data/edi_files_2/Synth00.edi'
savepath = r'C:/tmp/Occam1D_inv'

# make the save path if it doesn't exist
if not os.path.exists(savepath):
    os.mkdir(savepath)

# create data file
ocd = mtoc1d.Data() #create an object and assign values to arguments
ocd.write_data_file(edi_file=edi_file,
                    mode='det', # 'te', 'tm', or 'det'
                    save_path=savepath, # path to save to
                    res_errorfloor=4, # error floor in percentage
                    phase_errorfloor=2, # error floor in degrees
                    remove_outofquadrant=True)

# create model file
ocm = mtoc1d.Model(n_layers = 100, # number of layers
                  target_depth = 40000, # target depth in metres,
                                      # before padding
                  bottom_layer = 100000, # total depth of model
                  z1_layer=10 # first layer thickness in metres
                  )
ocm.write_model_file(save_path=savepath)

# create startup file
ocs = mtoc1d.Startup(data_fn=ocd.data_fn, # basename of data file
                    model_fn=ocm.model_fn, # basename for model file
                    max_iter=200, # maximum number of iterations to run
                    target_rms=1.0
                    )
ocs.write_startup_file()
```

Wrote Data File to : C:/tmp/Occam1D\_inv\Occam1d\_DataFile\_DET.dat

Wrote Model file: C:/tmp/Occam1D\_inv\Model1D

Wrote Input File: C:/tmp/Occam1D\_inv\OccamStartup1D

### 5.1.1 Viewing the outputs

Now that you have created the required inputs, you can run Occam1d on your files. Once you have done this, the following script can then be used to visualise the outputs as modelled and measured resistivity and phase, together with the 1D model. For this example below, we will use some example Occam 1D outputs from the mtpy examples directory.

```
In [2]: import mtpy.modeling.occam1d as mtoc1d
import os.path as op

# working directory
wd = r'C:/mtpywin/mtpy/examples/model_files/Occam1d'

# model and data file names
modelfn=op.join(wd, 'Model1D')
datafn=op.join(wd, 'Occam1d_DataFile_DET.dat')

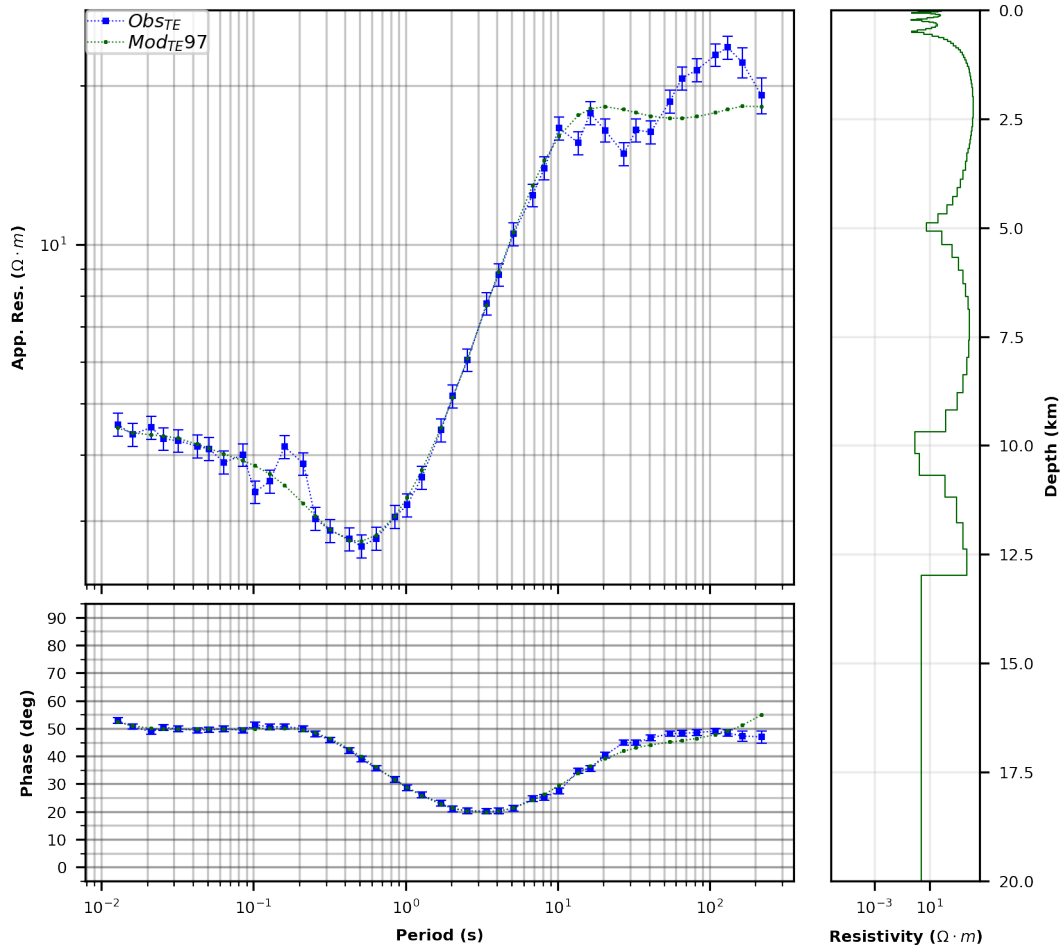
iterfn = op.join(wd, 'ITER_97.iter')
respfn = op.join(wd, 'ITER_97.resp')

# read in the model, don't need these lines to view the model
# but useful if you want to analyse the data
oc1m = mtoc1d.Model(model_fn=modelfn)
oc1m.read_iter_file(iterfn)

# read in the data file
oc1d = mtoc1d.Data(data_fn=datafn)
oc1d.read_data_file(data_fn=datafn)
oc1d.read_resp_file(resp_fn=respfn, data_fn=datafn)

# plot the model and response
pr = mtoc1d.Plot1DResponse(data_te_fn = datafn,
                           model_fn = modelfn,
                           resp_te_fn = respfn,
                           iter_te_fn = iterfn,
                           depth_limits = (0,20)
                           )
```

```
C:\mtpywin\Anaconda2\lib\site-packages\matplotlib\cbook\deprecation.py:107: MatplotlibDeprecationWarning:
warnings.warn(message, mplDeprecation, stacklevel=1)
```



## 5.2 Occam 2D

### 5.2.1 Creating inputs

The example below shows how to set up input files for modelling using the Occam 2D software (Constable et al. 1987, deGroot-Hedlin and Constable, 1990, <http://marineemlab.ucsd.edu/Projects/Occam/2DMT/index.html>).

For more details on the specifics of the Occam 2D inversion code and input options, please refer to the Occam 2D user manual (Occam2DMT\_FilesAndOptions.pdf) contained in the zip file from the above website. The input files required for modelling are described below.

There are four input files that are created.

- Data file - contains data for the inversions and relative station locations along the 2D profile used for the inversion
- Mesh file - contains mesh geometry information and information on which parameters are fixed/free



- Model file - details how mesh blocks are divided into model blocks (see Occam2D user manual)
- Startup file - contains control parameters for the inversion and the Data, Mesh, and Model file names

```
In [3]: import mtpy.modeling.occam2d_rewrite as occam2d
import os
import numpy as np

# path where edi files are located
edipath = r"C:/mtpywin/mtpy/examples/data/edi_files"

# path to save to
savepath = r'C:/tmp/Occam2D_inv'

# make the save path if it doesn't exist
if not os.path.exists(savepath):
    os.mkdir(savepath)

# list of stations
slst=[edi[0:-4] for edi in os.listdir(edipath) if edi.find('.edi')>0]

# create an occam data object
ocd = occam2d.Data(edi_path=edipath,
                  station_list=slst,
                  interpolate_freq=True, # interpolate frequencies
                  freq=np.logspace(-3,3,37) # frequency list to invert for
                  )
ocd.save_path = savepath

#### make data file
# geoelectric strike for rotation
# if not specified will calculate from the data
ocd.geoelectric_strike = 0.1

# error floors
ocd.res_te_err = 10 # error floor
ocd.res_tm_err = 10
ocd.phase_te_err = 5
ocd.phase_tm_err = 5
ocd.model_mode = '1' # modes to invert for - see
# https://mtpy2.readthedocs.io/en/develop/modeling.html#module-occam-2d
# for options
ocd.write_data_file()

# make model and mesh files
```

```

ocr = occam2d.Regularization(ocd.station_locations)
ocr.n_layers = 60 # number of layers
ocr.cell_width = 1250 # cell width to aim for, note this is the
                        # mesh size (2 mesh blocks across per model block)
ocr.x_pad_multiplier = 1.9 # controls size of padding
ocr.trigger= 0.25 # controls aspect ratio of blocks
ocr.z_bottom = 200000 # bottom of model
ocr.z1_layer = 50 # z1 layer and target depth in metres
ocr.z_target_depth = 80000 # target depth for inversion
ocr.save_path=ocd.save_path
ocr.build_mesh()
ocr.build_regularization()
ocr.write_mesh_file()
ocr.write_regularization_file()
#ocr.plot_mesh() # uncomment to plot the mesh

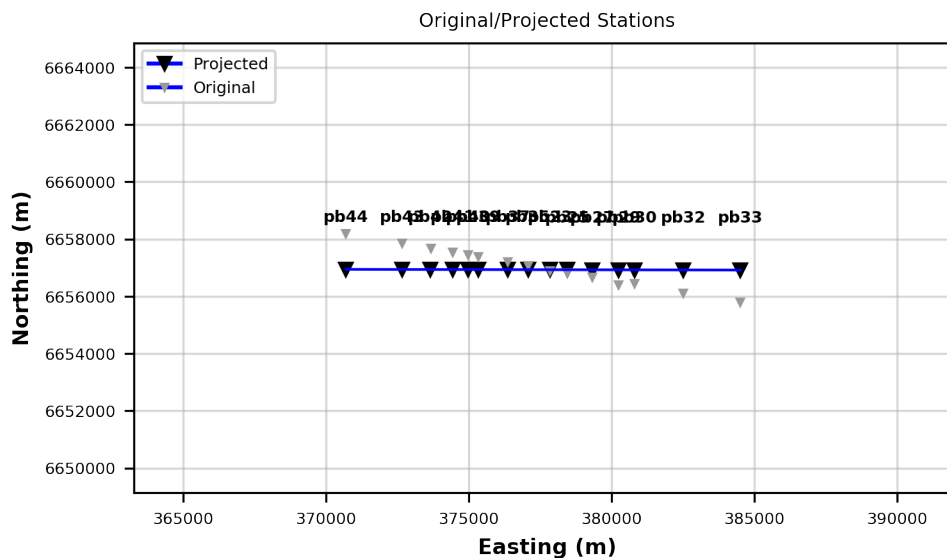
# make startup file
ocs=occam2d.Startup()
ocs.iterations_to_run=100
ocs.resistivity_start=2.0 # log10(starting resistivity)
ocs.target_misfit = 1.2
ocs.data_fn=op.join(ocd.data_fn)
ocr.get_num_free_params()
ocs.param_count=ocr.num_free_param
ocs.save_path=ocd.save_path
ocs.model_fn=ocr.reg_fn
ocs.write_startup_file()

```

```

=====
Rotated Z and Tipper to align with +0.10 degrees E of N
Profile angle is +90.10 degrees E of N
=====

```



Wrote Occam2D data file to C:/tmp/Occam2D\_inv\OccamDataFile.dat

#### MESH PARAMETERS

```

number of horizontal nodes = 160
number of vertical nodes   = 90
Total Horizontal Distance  = 28594.665131
Total Vertical Distance    = 293170.000000

```

#### REGULARIZATION PARAMETERS

```

binding offset      = 0.0
number layers       = 85
number of parameters = 4267
number of free param = 4267

```

#### MESH PARAMETERS

```

number of horizontal nodes = 33
number of vertical nodes   = 60
Total Horizontal Distance  = 904894.259656
Total Vertical Distance    = 358330.000000

```

#### REGULARIZATION PARAMETERS

```

binding offset      = 0.0
number layers       = 55
number of parameters = 498
number of free param = 498

```

```

=====
Wrote Mesh file to C:/tmp/Occam2D_inv\Occam2DMesh
Wrote Regularization file to C:/tmp/Occam2D_inv\Occam2DModel
Wrote Occam2D startup file to C:/tmp/Occam2D_inv\Occam2DStartup

```

```

C:\mtpywin\mtpy\mtpy\utils\calculator.py:322: RuntimeWarning: invalid value encountered in double
z_rel_err = error/z_amp

```

Now that you have created the required inputs, you can run Occam2d on your files. Once you have done this, the following script can then be used to visualise the outputs as modelled and measured resistivity and phase, together with the 2D model. For this example below, we use some example Occam 2D outputs from the mtpy examples directory.

## 5.2.2 Viewing the model

```

In [4]: # Import required modules
        from mtpy.modeling.occam2d_rewrite import PlotModel
        import os

        # path to directory containing inversion files
        idir = r'C:/mtpywin/mtpy/examples/model_files/Occam2d'
        savepath = r'C:/tmp'

        # iteration and response files
        iterfile = 'ITER12.iter'
        datafn = 'OccamDataFile.dat'

        # define file paths
        pltm = PlotModel(iter_fn=os.path.join(idir,iterfile),
                        data_fn=os.path.join(idir,datafn),
                        station_font_pad = 2.5, # pad (in plot units) between station
                                                # symbol and station label
                        station_font_size = 4, # station font size
                        station_font_rotation = 90, # rotation of the station font
                        font_size=4, # font size to use
                        climits=(0,4), # colour scale limits
                        cb_shrink= 0.3, # controls the size of the colorbar
                        xpad=0.5, # pad (in km) either side of profile
                        yscale = 'm', # 'm' or 'km'
                        yminorticks = 1.,
                        dpi=300, # resolution of figure
                        fig_aspect = 0.25, # aspect ratio between horizontal

```

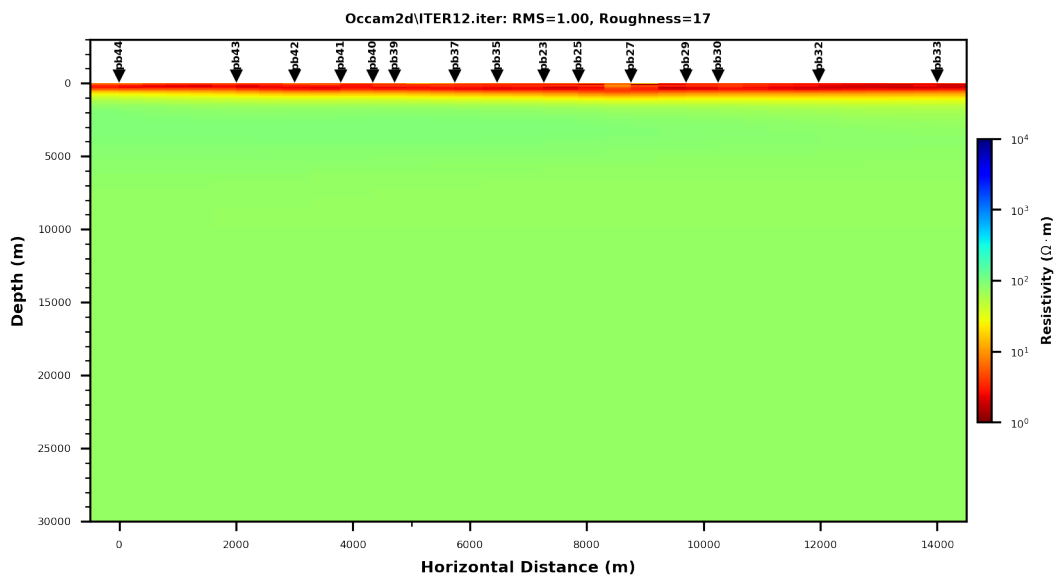
```

                                # and vertical scale
    ylimits = (-2,30), # depth limits
    stationid=(-1,3)) # index of station name to plot

    # save the model
    pltm.save_figure(os.path.join(savepath, 'model.png'))

Reading from C:/mtpywin/mtpy/examples/model_files/Occam2d\OccamDataFile.dat
    profile_angle = 97.0
    geoelectric_strike = 7.0
    number of sites = 15
    number of frequencies = 19

```



Saved figure to: C:/tmp\model.png

<Figure size 432x288 with 0 Axes>

### 5.2.3 Viewing the responses

Below is a simple example of plotting a response from a single station from an Occam 2D inversion. See the documentation pages for the Occam 2d mtpy modules for more details.

```

In [5]: # Import required modules
        from mtpy.modeling.occam2d_rewrite import PlotResponse
        import os

```

```

# path to directory containing inversion files
idir = r'C:/mtpywin/mtpy/examples/model_files/Occam2d'

# data and response files
datafn = 'OccamDataFile.dat'
respfile = 'RESP12.resp'

# plot response
pr = PlotResponse(os.path.join(idir,datafn),
                  resp_fn = os.path.join(idir,respfile),
                  plot_type=['pb40'] # either a list of stations to plot
                                # or '1' to plot all stations
                  )
pr.save_figures(os.path.join(savepath,'Occam2dresponses'),
               fig_fmt='png')

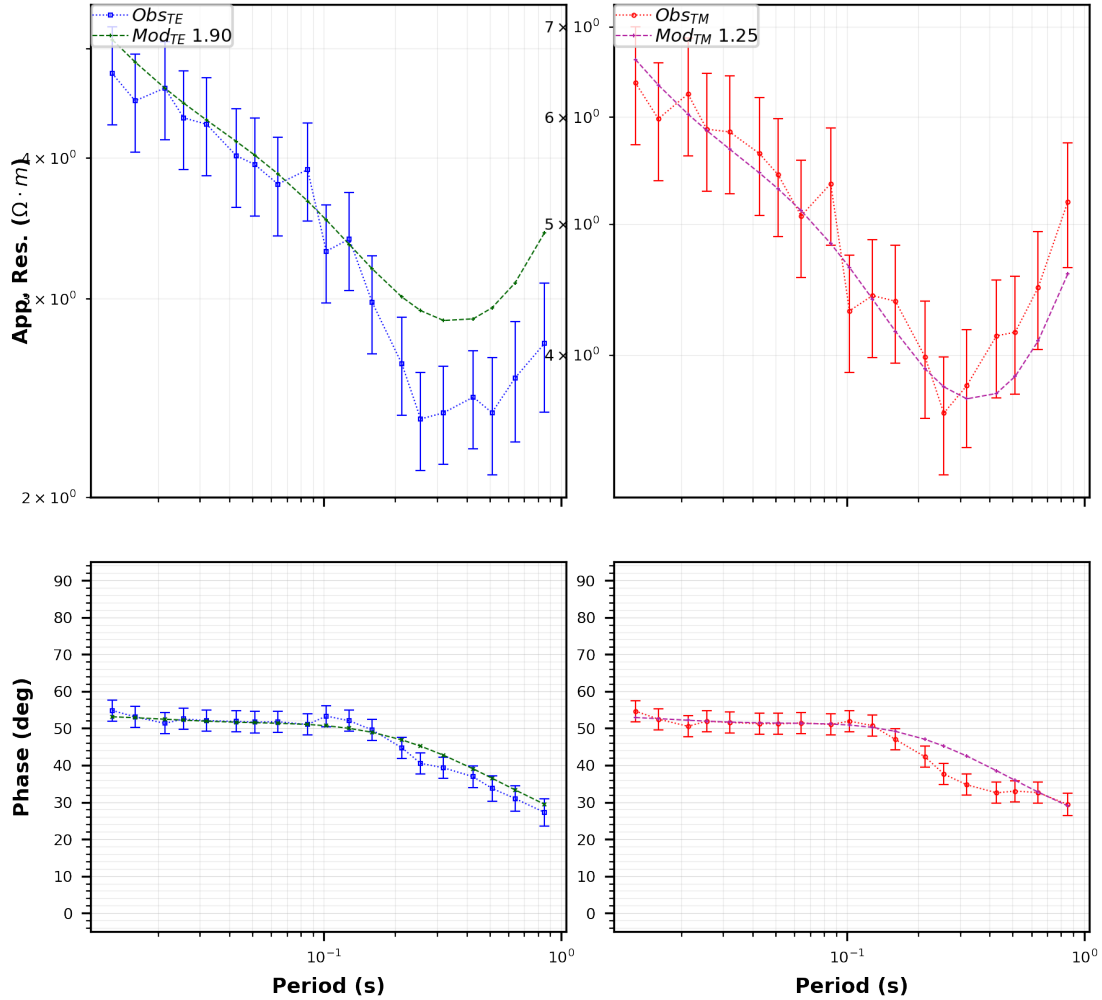
```

```

Reading from C:/mtpywin/mtpy/examples/model_files/Occam2d\OccamDataFile.dat
profile_angle = 97.0
geoelectric_strike = 7.0
number of sites = 15
number of frequencies = 19

```

pb40



saved figure to C:/tmp\Occam2dresponses\pb40\_resp.png

## 5.3 ModEM

### 5.3.1 Creating Inputs

The example below demonstrates how to set up input files for 3D inversion using the ModEM inversion code (Egbert and Kelbert 2012; Kelbert et al. 2014; <http://blogs.oregonstate.edu/modem3dmt/>).

The ModEM code has three compulsory input files:

- Model file - contains information on the mesh size and starting resistivity
- Data file - contains the impedance tensor and tipper data at each frequency and station locations

- Covariance file - contains model covariance parameters, which control smoothing

**Data file** The data file needs to be created before any of the other files can be made. This is because the data file reads the station locations, and these are then used to create a grid centred on the stations. There are a number of different parameters you can vary - see the comments next to each line when creating the data object. You need to check these are correct - MTPy assumes you know what you are doing and will not give you an error if you choose the wrong projection, for example.

```
In [6]: import os
        from mtpy.modeling.modem import Data
        from mtpy.core.edi import Edi
        from mtpy.utils.calculator import get_period_list
        import numpy as np

        # path to save to
        workdir = r'C:/tmp/ModEM_inv'

        # make the save path if it doesn't exist
        if not os.path.exists(workdir):
            os.mkdir(workdir)

        # path containing edi files
        edipath = r'C:/mtpywin/mtpy/examples/data/edi_files_2'

        # find all files in edipath
        edi_list = [os.path.join(edipath,ff) for ff in os.listdir(edipath) if \
                    (ff.endswith('.edi'))]

        # define period list to interpolate data onto
        # MTPy will not extrapolate outside the data period range
        # this code gets 4 periods per decade
        period_list = get_period_list(1e-4,3e3,4)

        # create a data object
        do = Data(edi_list=edi_list,
                  inv_mode = '1', # invert for Z + tipper ('2' means Z only)
                  save_path=workdir,
                  period_list=period_list,
                  period_buffer=2., # buffer factor to determine how far to interpolate
                                   # between points. 2 means interpolate by a factor of
                                   # 2. e.g. if the data is at 10s the code will
                                   # interpolate only from 5 to 20 s.
                  error_type_z='floor_egbert', # error type (egbert is % of sqrt(zxy*zyx))
                  error_value_z=5., # error floor in percent
                  error_type_tipper = 'floor_abs', # type of error to set in tipper,

                                   # floor_abs is an absolute value set
```



```

                                # as a floor
    error_value_tipper=0.01,
    model_epsg=28354 # model epsg, currently set to utm zone 54.
                    # See http://spatialreference.org/ to
                    # find the epsg code for your projection
)
# write data file, setting elevation to zero as we haven't added topography
do.write_data_file()

# save epsg code and centre position to a text file
np.savetxt(os.path.join(workdir, 'center_position.txt'),
            np.array([do.center_point['east'],
                      do.center_point['north']]))
np.savetxt(os.path.join(workdir, 'epsg.txt'), [do.model_epsg])

```

If you want to write a vtk file for 3d viewing, you need download and install evtk from <https://>

Note: if you are using Windows you should build evtk first with either MinGW or cygwin using the

```
python setup.py build -compiler=mingw32 or
```

```
python setup.py build -compiler=cygwin
```

If you want to write a vtk file for 3d viewing, you need to install pyevtk

Note: if you are using Windows you should build evtk first with either MinGW or cygwin using the

```
python setup.py build -compiler=mingw32 or
```

```
python setup.py build -compiler=cygwin
```

If you want to write a vtk file for 3d viewing, you need download and install evtk from <https://>

If you want to write a vtk file for 3d viewing, you need to pip install PyEVTK: <https://bitbucket.org/pyevtk/>

Note: if you are using Windows you should build evtk first with either MinGW or cygwin using the

```
python setup.py build -compiler=mingw32 or
```

```
python setup.py build -compiler=cygwin
```

If you want to write a vtk file for 3d viewing, you need download and install evtk from <https://>

-----  
Inverting for periods:

```

0.000100
0.000178
0.000316
0.000562
0.001000
0.001778
0.003162
0.005623
0.010000
0.017783
0.031623
0.056234
0.100000

```

```
0.177828
0.316228
0.562341
1.000000
1.778279
3.162278
5.623413
10.000000
17.782794
31.622777
56.234133
100.000000
177.827941
316.227766
562.341325
1000.000000
1778.279410
3162.277660
```

-----  
2018-10-05T13:55:58 - Data - INFO - Wrote ModEM data file to C:/tmp/ModEM\_inv\ModEM\_Data.dat

**Model file** Once the data file has been created, we now create the model file. MTPy creates a grid that covers the stations with the provided cell size. You can update parameters like cell size, target depth, padding parameters, and first layer thickness as described below.

Once the model is created, we then add topography. In the example below we will use etopo1 topography/bathymetry data downloaded from <https://www.ngdc.noaa.gov/mgg/global/>.

After topography is added to the model, we then need to change the station elevations so that the stations sit on the topography.

```
In [7]: from mtpy.modeling.modem import Model
```

```
# create a Model object
mo = Model(station_locations=do.station_locations,
            cell_size_east=8000,
            cell_size_north=8000,
            pad_north=7, # number of padding cells N and S
            pad_east=7, # number of padding cells E and W
            pad_z=6, # number of vertical padding cells
            pad_num=6, # number of cells outside station area before padding
            pad_stretch_v=1.6, # increase factor in padding cells (vertical)
            pad_stretch_h=1.9, # increase factor in padding cells (horizontal)
            n_air_layers = 10, # number of air layers
            res_initial_value=100, # halfspace resistivity value
                                # for reference model (ohm-m)
            n_layers=100, # total number of z layers, including air
            z1_layer=50, # first layer thickness
            pad_method='stretch', # method for calculating padding
```

```

        z_target_depth=300000 # approx. depth to bottom of core model
                                # (padding after this depth)
    )

    mo.make_mesh()
    mo.write_model_file(save_path=workdir)

    ## add topography to the model and rewrite
    topo_file = r'C:/mtpywin/mtpy/examples/data/AussieContinent_etopo1.asc'
    mo.add_topography_to_model2(topo_file)
    mo.write_model_file(save_path=workdir)

    # project the stations on the topography
    do.project_stations_on_topography(mo)

-----
    Number of stations = 28
    Dimensions:
        e-w = 87
        n-s = 73
        z = 101 (without 7 air layers)
    Extensions:
        e-w = 2147600.0 (m)
        n-s = 2035600.0 (m)
        0-z = 1188110.0 (m)
    Stations rotated by: 0.0 deg clockwise positive from N

** Note ModEM does not accommodate mesh rotations, it assumes
    all coordinates are aligned to geographic N, E
    therefore rotating the stations will have a similar effect
    as rotating the mesh.

-----
    Elevation data type and shape *** <type 'numpy.ndarray'> (72L, 86L) 72 72
-----

Inverting for periods:
    0.000100
    0.000178
    0.000316
    0.000562
    0.001000
    0.001778
    0.003162
    0.005623
    0.010000
    0.017783
    0.031623
    0.056234
    0.100000

```

0.177828  
0.316228  
0.562341  
1.000000  
1.778279  
3.162278  
5.623413  
10.000000  
17.782794  
31.622777  
56.234133  
100.000000  
177.827941  
316.227766  
562.341325  
1000.000000  
1778.279410  
3162.277660

-----  
2018-10-05T13:56:13 - Data - INFO - Wrote ModEM data file to C:/tmp/ModEM\_inv\ModEM\_Data.dat

Out[7]: ([16,  
18,  
20,  
22,  
24,  
26,  
28,  
30,  
32,  
34,  
36,  
38,  
40,  
42,  
44,  
46,  
48,  
50,  
52,  
54,  
55,  
57,  
59,  
61,  
63,  
65,

```

67,
69],
[55,
53,
52,
50,
49,
47,
46,
44,
43,
41,
40,
38,
37,
35,
34,
32,
31,
29,
28,
26,
25,
23,
22,
20,
19,
17,
16,
14])

```

The final step is to create a covariance file, which provides information on the location of free vs fixed parameters, and smoothing. The covariance file is created based on the model file we generated in the previous step.

```
In [8]: from mtpy.modeling.modem import Covariance
```

```

# create a covariance file
co = Covariance()
co.smoothing_east = 0.6
co.smoothing_north = 0.6
co.smoothing_z = 0.6
co.write_covariance_file(model_fn=mo.model_fn)

```

```

-1017800.000    -1073800.000    -473.000

```

```

0.000

```

Reading C:/tmp/ModEM\_inv\ModEM\_Model\_File.rho

### 5.3.2 Viewing outputs

Once you've run ModEM there are quite a few functions available to look at model outputs. Alternatively, you can use MTPy to write the model to a Gocad sgrid format and use Gocad to interrogate the model.

**Resistivity model slice** The first function we demonstrate is how to plot a ModEM depth slice. Note that this function has an option for interactive plotting if you run it as a script and set `plot_yn` to 'y'. Here we will just use static plotting.

```
In [9]: import os
        from mtpy.modeling.modem import PlotSlices

        wd = r'C:/mtpywin/mtpy/examples/model_files/ModEM_2'

        savepath = r'C:/tmp'

        model_fn = os.path.join(wd, 'Modular_MPI_NLCG_004.rho')
        data_fn = os.path.join(wd, 'ModEM_Data.dat')

        p0bj = PlotSlices(model_fn=model_fn,
                           data_fn=data_fn,
                           save_path=wd,
                           #   climits = [0,2], # log10(colour limits)
                           cmap='bwr_r', # colour map
                           plot_stations=True, # True/False - show station locations or not
                           station_id=[5,8], # indices (start,finish) of station label to plot
                           #   ew_limits=[-220,220], # east-west limits, if not provided
                           #                                     # will auto calculate from data
                           #   ns_limits=[-170,170], # north-south limits, if not provided
                           #                                     # will auto calculate from data
                           font_size=6, # font size on plots
                           fig_size=(6,3), # figure size
                           plot_yn='n', # whether to load interactive plotting
                           fig_dpi = 400 # change to your preferred file resolution
                           )

        p0bj.save_path = savepath
        p0bj.export_slices(plane='N-E', # options are 'N-Z', 'E-Z', and 'N-E'
                           indexlist=[32], # slice indices to plot
                           station_buffer=20e3, # distance threshold for plotting
                                                # stations on vertical slice
                           save=True,
                           )
```

```

-386800.000    -442800.000    0.000

0.000

```

2018-10-05T13:56:14 - Data - INFO - Set rotation angle to 0.0 deg clockwise from N

```

=====
Buttons      Description
=====
'e'          moves n-s slice east by one model block
'w'          moves n-s slice west by one model block
'n'          moves e-w slice north by one model block
'm'          moves e-w slice south by one model block
'd'          moves depth slice down by one model block
'u'          moves depth slice up by one model block
=====

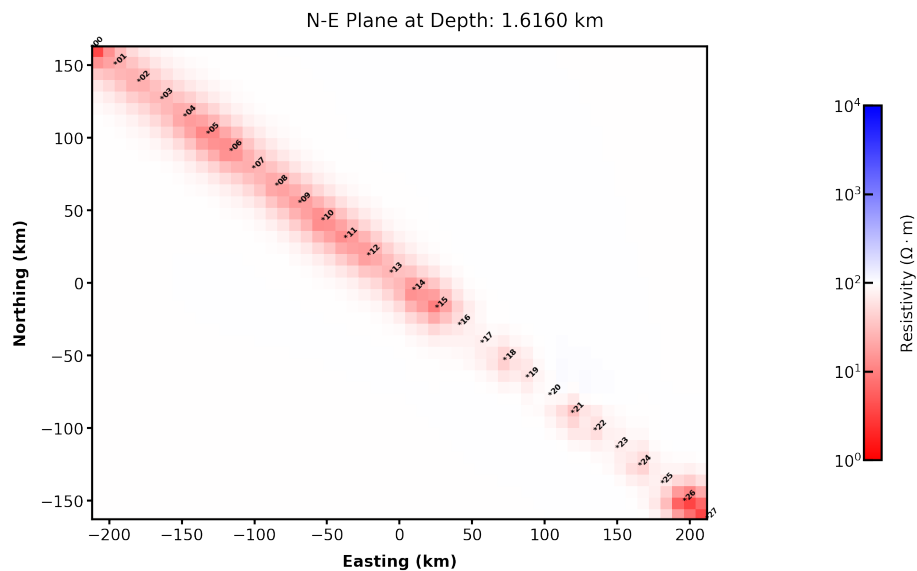
```

Exporting C:/tmp\N-E-plane-at-Depth032.1.616.km.png..

```

Out[9]: ([<Figure size 2400x1200 with 2 Axes>],
         ['C:/tmp\\N-E-plane-at-Depth032.1.616.km.png'])

```



**Plot model along arbitrary profile** The example below shows how to plot stations along an arbitrary profile, which is particularly useful in the case where you may have run a 3D model on 2D profile data. Please see the code below for parameters that can be varied.

```

In [10]: import os

import matplotlib.pyplot as plt

```

```

from matplotlib import colors
import numpy as np

from mtpy.modeling.modem import PlotSlices

# working directory and save path
wd = r'C:/mtpywin/mtpy/examples/model_files/ModEM_2'
savepath = r'C:/tmp'

# model and data file names
model_fn = os.path.join(wd, 'Modular_MPI_NLCG_004.rho')
data_fn = os.path.join(wd, 'ModEM_Data.dat')
fs = 8 # fontsize on plot

# create a PlotSlices object
ps = PlotSlices(model_fn=model_fn,
                 data_fn=data_fn,
                 save_path=wd,
                 plot_yn='n')

# create a new figure
fig = plt.figure(figsize=[6,3])
ax = plt.subplot(111)

# get resistivity data along a slice defined by the stations
gd, gz, gv = ps.get_slice("STA",
                           nsteps=200 # total number of cells across
                           )

## to get resistivity along an arbitrary profile
## (in this case stations 5 to 9)
## xy locations
#coords = np.vstack([ps.station_east[5:10],ps.station_north[5:10]]).T
#gd, gz, gv = ps.get_slice("XY", nsteps=50, coords=coords)

ci = ax.pcolormesh(gd,gz, gv,
                   vmin=1,vmax=1e4, # min & max resistivity on colour map
                   norm=colors.LogNorm(), # plot colours on a log scale
                   cmap='bwr_r', # colour map
                   rasterized=True)

# set some plot parameters and add labels
ax.invert_yaxis()
ax.set_aspect(1)
ax.set_ylim(1e2)
plt.setp(ax.get_xticklabels(),fontsize=fs)
plt.setp(ax.get_yticklabels(),fontsize=fs)
plt.xlabel('Distance, km',fontsize=fs)
plt.ylabel('Depth, km',fontsize=fs)

```



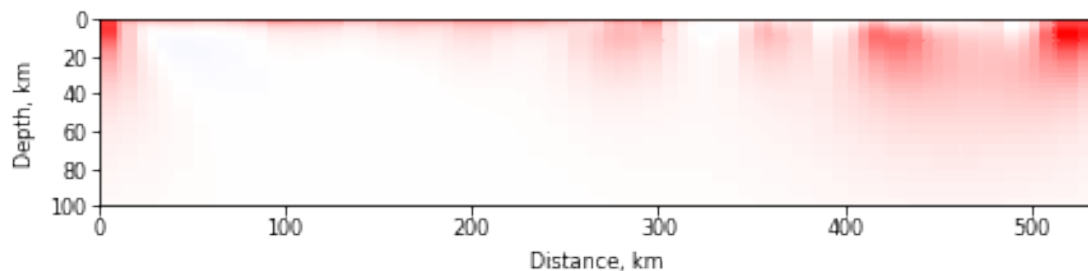
```
plt.savefig(os.path.join(savepath, 'DepthSlice.png'),
            dpi=400) # change to your desired figure resolution

-386800.000    -442800.000    0.000

0.000
```

2018-10-05T13:56:18 - Data - INFO - Set rotation angle to 0.0 deg clockwise from N

Buttons	Description
'e'	moves n-s slice east by one model block
'w'	moves n-s slice west by one model block
'n'	moves e-w slice north by one model block
'm'	moves e-w slice south by one model block
'd'	moves depth slice down by one model block
'u'	moves depth slice up by one model block



**Data and model response** The example below shows how to plot the data together with the model response at one or more stations.

```
In [11]: import os
          from mtpy.modeling.modem import PlotResponse

          # working directory and save path
          wd = r'C:/mtpywin/mtpy/examples/model_files/ModEM_2'
          savepath = r'C:/tmp'

          # response and data files
          respfn = 'Modular_MPI_NLCG_004.dat'
          datafn = 'ModEM_Data.dat'

          # make a plot object
```

```

ro = PlotResponse(data_fn=os.path.join(wd,datafn),
                  resp_fn=os.path.join(wd,respfn),
                  plot_type=['Synth02'], # provide a list of station names
                                      # or '1' to plot all stations
                  plot_style=1, # 1 for 4-columns; 2 for 2-columns
                  plot_z=False,
                  )

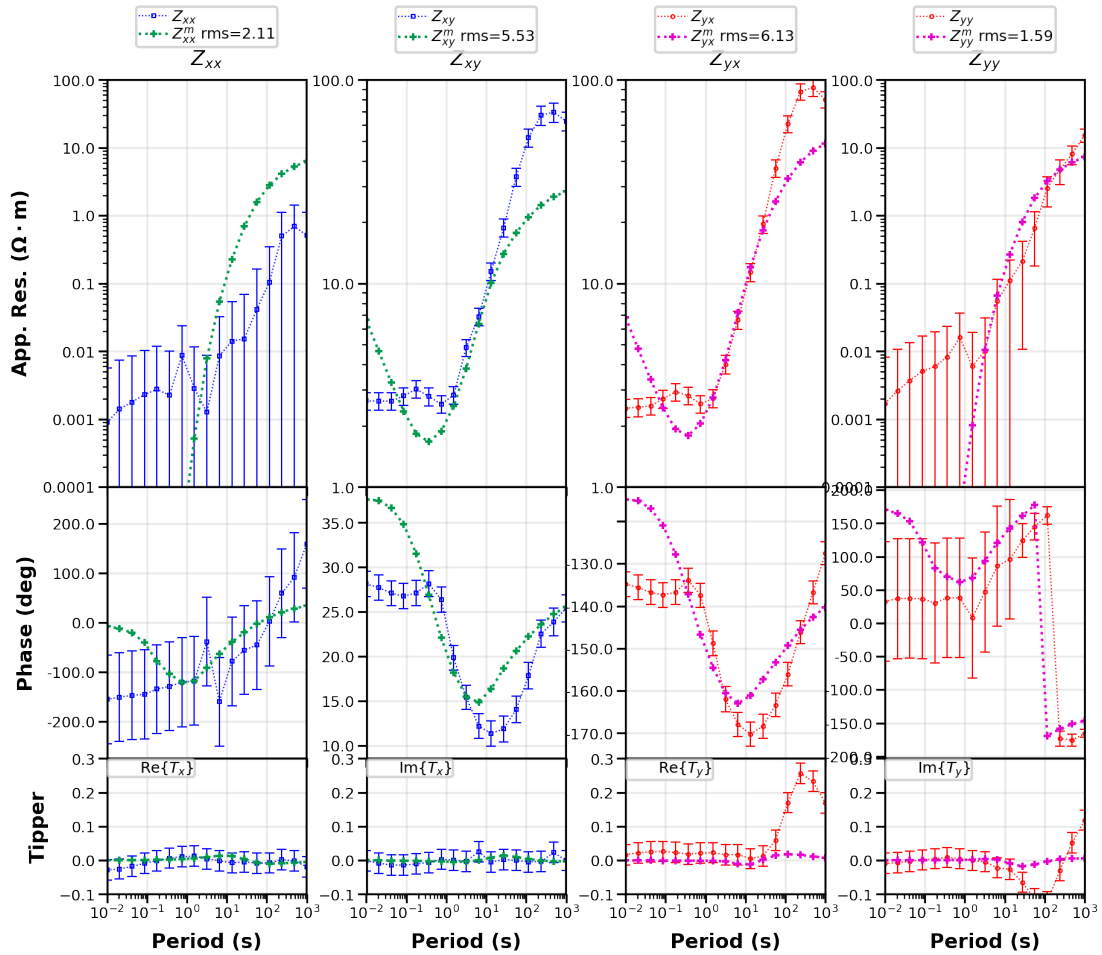
ro.plot()

ro.save_figure(os.path.join(savepath,'response.png'),
              fig_dpi=400) # change fig_dpi to your desired resolution

```

2018-10-05T13:56:22 - Data - INFO - Set rotation angle to 0.0 deg clockwise from N  
2018-10-05T13:56:22 - Data - INFO - Set rotation angle to 0.0 deg clockwise from N  
Plotting: Synth02  
self.plot\_tipper = True

Synth02



Saved figure to: C:/tmp/response.png

**Phase tensor plots** This example shows how to plot model and response phase tensors for a specified frequency, as a map. Depending on what inputs you provide, you can plot the data, model response, and the residual phase tensor, which is a measure of the difference between the data and model response.

```
In [12]: import os
         from mtpy.modeling.modem.phase_tensor_maps import PlotPTMaps

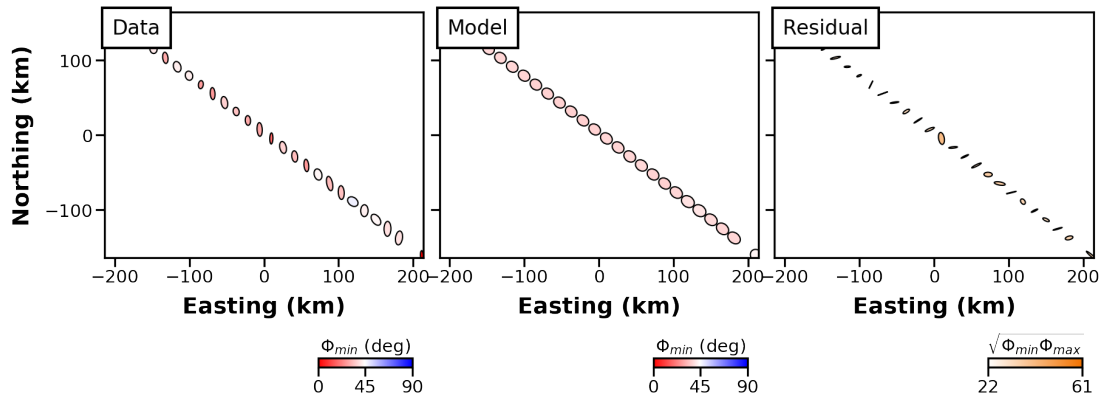
workdir = r'C:/mtpywin/mtpy/examples/model_files/ModEM_2'
savepath= r'C:/tmp'

ptmap = PlotPTMaps(data_fn=os.path.join(workdir, 'ModEM_Data.dat'),
                    resp_fn=os.path.join(workdir, 'Modular_MPI_NLCG_004.dat'),
                    cb_pt_pad=0.0, # y position of colorbar on plot
                    fig_size=(6,2.5),
                    ellipse_dict = {'size': 20,
                                   'ellipse_range':[0, 90],
                                   'ellipse_colorby':'phimin',
                                   'ellipse_cmap':'bwr_r'}, #mt_bl2gr2rd
                    residual_cmap='mt_wh2or'
                    )

for period_index in [16]: # customise which periods to plot
    ptmap.plot(period=period_index, # index of period to plot
               edgecolor='k', # colour for edge of ellipses
               lw = 0.5 # linewidth of edge of ellipses
               )

# save all plots to file
ptmap.save_figure(save_path=savepath,
                  file_format='png',
                  fig_dpi=400) # change to your desired resolution

('The input parameter period is', 16)
2018-10-05T13:56:27 - Data - INFO - Set rotation angle to 0.0 deg clockwise from N
2018-10-05T13:56:27 - Data - INFO - Set rotation angle to 0.0 deg clockwise from N
Plotting Period: 1000
{'edgecolor': 'k', 'lw': 0.5}
```



Saved figure to: C:/tmp\PT\_DepthSlice\_images.png

**RMS maps** These functions plot the (normalised) data-model misfit. You can choose an integer value for `period_index` to plot a particular period, or just specify 'all' to plot the root-mean-square misfit over all periods. The plots are broken up according to component -  $Z_{XX}$ ,  $Z_{XY}$ ,  $Z_{YX}$ ,  $Z_{YY}$ ,  $T_X$ ,  $T_Y$ .

```
In [13]: import os
import numpy as np
from mtpy.modeling.modem import PlotRMSMaps

wd = r'C:/mtpywin/mtpy/examples/model_files/ModEM_2'
savepath = r'C:/tmp'

resid_fn = os.path.join(wd, 'Modular_MPI_NLCG_004.res')

# create a RMS plot object
probj = PlotRMSMaps(resid_fn,
                    rms_min=1, # min rms for colorbar
                    rms_max=10, # max rms for colorbar
                    period_index='all' # 'all' or integer value
                    )

# save figure
probj.save_figure(save_path=savepath,
                  fig_close=False, # whether or not to close figure
                  save_fig_dpi = 400 # change to your preferred figure resolution
                  )
```

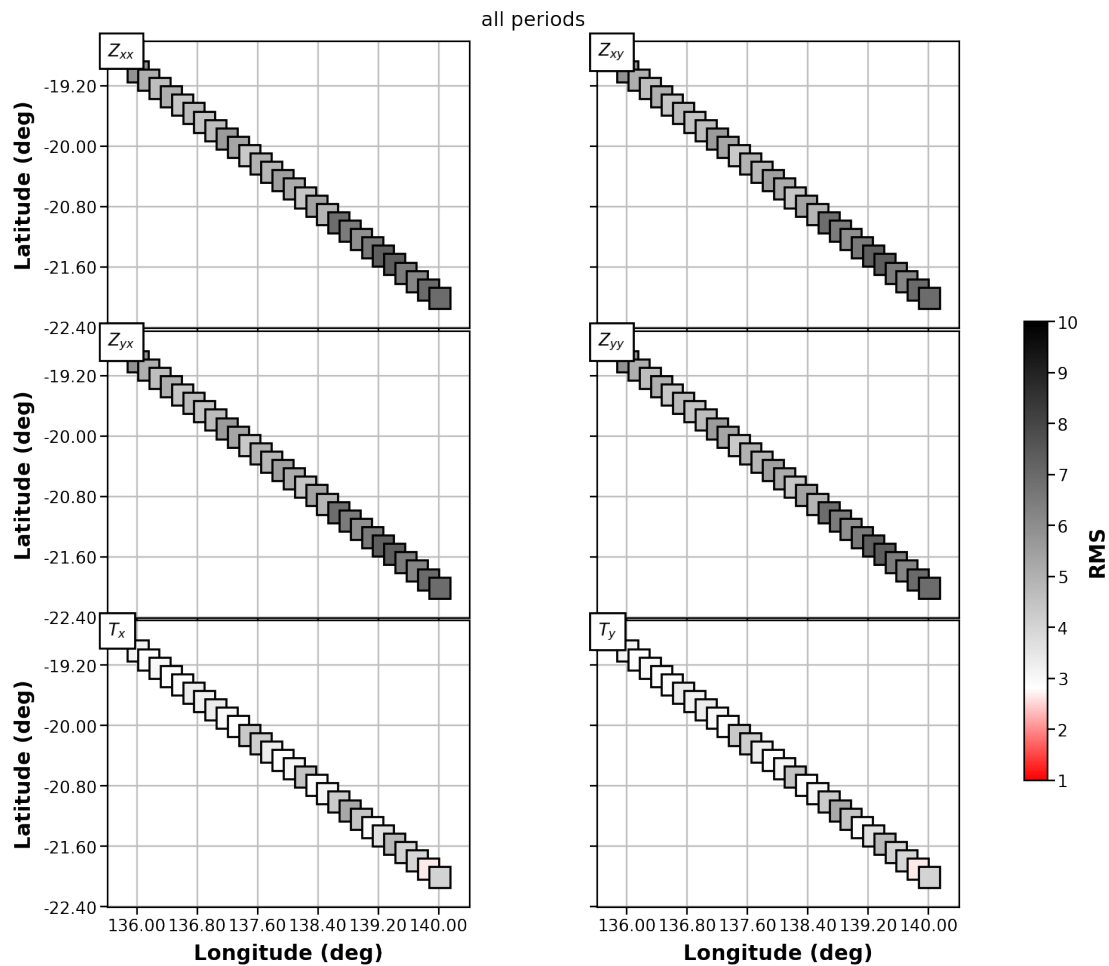
2018-10-05T13:56:30 - Data - INFO - Set rotation angle to 0.0 deg clockwise from N

```

C:\mtpywin\mtpy\mtpy\modeling\modem\residual.py:197: RuntimeWarning: invalid value encountered i
    z_norm = np.abs(res_vals['z']) / (np.real(res_vals['z_err']) * 2. ** 0.5)
C:\mtpywin\mtpy\mtpy\modeling\modem\residual.py:211: RuntimeWarning: invalid value encountered i
    tip_norm = np.abs(res_vals['tip']) / (np.real(res_vals['tip_err']) * 2. ** 0.5)
C:\mtpywin\Anaconda2\lib\site-packages\matplotlib\figure.py:457: UserWarning: matplotlib is curr
    "matplotlib is currently using a non-GUI backend, "

```

saved file to C:/tmp\RMS\_AllPeriods.png



### 5.3.3 Conversions

**Gocad sgrid format** For advanced analysis, you may want to convert a ModEM model into a different format for analysis in another software package. In MTPy, we support export to Gocad sgrid format. The resulting sgrid has a resistivity property, which has a logarithmic colour stretch applied to it when it is imported into Gocad.

To export the model to Gocad you have the option of providing the centre position of your model in projected coordinates (metres). If this is not provided, your model will appear in relative grid coordinates when you bring it into Gocad.

```
In [14]: from mtpy.modeling.modem import Model
import os

wd = r'C:/mtpywin/mtpy/examples/model_files/ModEM_2'
savepath = r'C:/tmp'

# filename of the model file to export
iterfn = os.path.join(wd, 'Modular_MPI_NLCG_004.rho')

# if you like, provide centre position of model in real
# world coordinates (eastings/northings)
centre = np.array([0., 0., 0.])

# read the model in
mObj = Model(model_fn=os.path.join(wd, iterfn))
mObj.read_model_file()

# write an sgrid file
mObj.write_gocad_sgrid_file(origin=centre,
                             clip=[0,0,0]) # number of cells to clip off the
                                             # model in x (east-west), y (north-
                                             # south) and z directions.

-386800.000      -442800.000      0.000

0.000
```