# mtpy Documentation

***Release 1.01.01***

**mtgeophysics**

**Sep 20, 2018**

# Contents

Contents:

# CHAPTER 1

# Core:

## 1.1 Module z

**exception** mtpy.core.z.**MT_Z_Error**

**class** mtpy.core.z.**ResPhase**(*z_array=None*, *z_err_array=None*, *freq=None*, ***kwargs*)
resistivity and phase container

**compute_resistivity_phase**(*z_array=None*, *z_err_array=None*, *freq=None*)
compute resistivity and phase from z and z_err

**set_res_phase**(*res_array*, *phase_array*, *freq*, *res_err_array=None*, *phase_err_array=None*)
Set values for resistivity (res - in Ohm m) and phase (phase - in degrees), including error propagation.

**Parameters**

- **res_array** (*np.ndarray (num_freq, 2, 2)*) – resistivity array in Ohm-m
- **phase_array** (*np.ndarray (num_freq, 2, 2)*) – phase array in degrees
- **freq** (*np.ndarray (num_freq)*) – frequency array in Hz
- **res_err_array** (*np.ndarray (num_freq, 2, 2)*) – resistivity error array in Ohm-m
- **phase_err_array** (*np.ndarray (num_freq, 2, 2)*) – phase error array in degrees

**class** mtpy.core.z.**Tipper**(*tipper_array=None*, *tipper_err_array=None*, *freq=None*)
Tipper class –> generates a Tipper-object.

Errors are given as standard deviations (sqrt(VAR))

**Parameters**

- **tipper_array** (*np.ndarray ((nf, 1, 2), dtype='complex')*) – tipper array in the shape of [Tx, Ty] *default* is None

- **tipper_err_array** (*np.ndarray((nf, 1, 2))*) – array of estimated tipper errors in the shape of [Tx, Ty]. Must be the same shape as tipper_array. *default* is None
- **freq** (*np.ndarray(nf)*) – array of frequencies corresponding to the tipper elements. Must be same length as tipper_array. *default* is None

| Attributes | Description |
|---|---|
| freq | array of frequencies corresponding to elements of z |
| rotation_angle | angle of which data is rotated by |
| tipper | tipper array |
| tipper_err | tipper error array |

| Methods | Description |
|---|---|
| mag_direction | computes magnitude and direction of real and imaginary induction arrows. |
| amp_phase | computes amplitude and phase of Tx and Ty. |
| rotate | rotates the data by the given angle |

**compute_amp_phase**()

**Sets attributes:**

- *amplitude*
- *phase*
- *amplitude_err*
- *phase_err*

values for resistivity are in in Ohm m and phase in degrees.

**compute_mag_direction**()
computes the magnitude and direction of the real and imaginary induction vectors.

**rotate**(*alpha*)
Rotate Tipper array.

Rotation angle must be given in degrees. All angles are referenced to geographic North=0, positive in clockwise direction. (Mathematically negative!)

In non-rotated state, 'X' refs to North and 'Y' to East direction.

**Updates the attributes:**

- *tipper*
- *tipper_err*
- *rotation_angle*

**set_amp_phase**(*r_array*, *phi_array*)
Set values for amplitude(r) and argument (phi - in degrees).

**Updates the attributes:**

- tipper
- tipper_err

**set_mag_direction**(*mag_real*, *ang_real*, *mag_imag*, *ang_imag*)
computes the tipper from the magnitude and direction of the real and imaginary components.

Updates tipper

No error propagation yet

**class** `mtpy.core.z.Z`(*z_array=None*, *z_err_array=None*, *freq=None*)

Z class - generates an impedance tensor (Z) object.

Z is a complex array of the form (n_freq, 2, 2), with indices in the following order:

- Zxx: (0,0)
- Zxy: (0,1)
- Zyx: (1,0)
- Zyy: (1,1)

All errors are given as standard deviations (sqrt(VAR))

**Parameters**

- **z_array** (*numpy.ndarray(n_freq, 2, 2)*) – array containing complex impedance values
- **z_err_array** (*numpy.ndarray(n_freq, 2, 2)*) – array containing error values (standard deviation) of impedance tensor elements
- **freq** (*np.ndarray(n_freq)*) – array of frequency values corresponding to impedance tensor elements.

| Attributes | Description |
|---|---|
| freq | array of frequencies corresponding to elements of z |
| rotation_angle | angle of which data is rotated by |
| z | impedance tensor |
| z_err | estimated errors of impedance tensor |
| resistivity | apparent resisitivity estimated from z in Ohm-m |
| resistivity_err | apparent resisitivity error |
| phase | impedance phase (deg) |
| phase_err | error in impedance phase |

| Methods | Description |
|---|---|
| det | calculates determinant of z with errors |
| invariants | calculates the invariants of z |
| inverse | calculates the inverse of z |
| re-move_distortion | removes distortion given a distortion matrix |
| remove_ss | removes static shift by assumin Z = S * Z_0 |
| norm | calculates the norm of Z |
| only1d | zeros diagonal components and computes the absolute valued mean of the off-diagonal components. |
| only2d | zeros diagonal components |
| res_phase | computes resistivity and phase |
| rotate | rotates z positive clockwise, angle assumes North is 0. |
| set_res_phase | recalculates z and z_err, needs attribute freq |
| skew | calculates the invariant skew (off diagonal trace) |
| trace | calculates the trace of z |

**Example**

```
>>> import mtpy.core.z as mtz
>>> import numpy as np
>>> z_test = np.array([[0+0j, 1+1j], [-1-1j, 0+0j]])
>>> z_object = mtz.Z(z_array=z_test, freq=[1])
>>> z_object.rotate(45)
>>> z_object.resistivity
```

**det**
Return the determinant of Z

**Returns** det_Z

**Return type** np.ndarray(nfreq)

**det_err**
Return the determinant of Z error

**Returns** det_Z_err

**Return type** np.ndarray(nfreq)

**freq**
Frequencies for each impedance tensor element

Units are Hz.

**invariants**
Return a dictionary of Z-invariants.

**Contains**

- z1
- det
- det_real
- det_imag
- trace
- skew
- norm
- lambda_plus/minus,
- sigma_plus/minus

**inverse**
Return the inverse of Z.

(no error propagtaion included yet)

**norm**
Return the 2-/Frobenius-norm of Z

**Returns** norm

**Return type** np.ndarray(nfreq)

**norm_err**
Return the 2-/Frobenius-norm of Z error

**Returns** norm_err

**Return type** np.ndarray(nfreq)

**only_1d**
Return Z in 1D form.

If Z is not 1D per se, the diagonal elements are set to zero, the off-diagonal elements keep their signs, but their absolute is set to the mean of the original Z off-diagonal absolutes.

**only_2d**
Return Z in 2D form.

If Z is not 2D per se, the diagonal elements are set to zero.

**remove_distortion**(*distortion_tensor*, *distortion_err_tensor=None*)
Remove distortion D form an observed impedance tensor Z to obtain the uperturbed "correct" Z0: Z = D * Z0

Propagation of errors/uncertainties included

**Parameters**

- **distortion_tensor** (*np.ndarray(2, 2, dtype=real)*) – real distortion tensor as a 2x2
- **distortion_err_tensor** – default is None

**Return type**

np.ndarray(2, 2, dtype='real')

**returns** impedance tensor with distorion removed

**Return type**

np.ndarray(num_freq, 2, 2, dtype='complex')

**returns** impedance tensor error after distortion is removed

**Return type**

np.ndarray(num_freq, 2, 2, dtype='complex')

**Example**

```
>>> import mtpy.core.z as mtz
>>> distortion = np.array([[1.2, .5],[.35, 2.1]])
>>> d, new_z, new_z_err = z_obj.remove_distortion(distortion)
```

**remove_ss**(*reduce_res_factor_x=1.0*, *reduce_res_factor_y=1.0*)
Remove the static shift by providing the respective correction factors for the resistivity in the x and y components. (Factors can be determined by using the "Analysis" module for the impedance tensor)

Assume the original observed tensor Z is built by a static shift S and an unperturbated "correct" Z0 :

- Z = S * Z0

**therefore the correct Z will be :**

- Z0 = S^(-1) * Z

**Parameters**

- **reduce_res_factor_x** (*float or iterable list or array*) – static shift factor to be applied to x components (ie z[:, 0, :]). This is assumed to be in resistivity scale

- **reduce_res_factor_y** (*float or iterable list or array*) – static shift factor to be applied to y components (ie z[:, 1, :]). This is assumed to be in resistivity scale

**Returns** static shift matrix,

**Return type** np.ndarray ((2, 2))

**Returns** corrected Z

**Return type** *mtpy.core.z.Z*

**Note:** The factors are in resistivity scale, so the entries of the matrix "S" need to be given by their square-roots!

**rotate** (*alpha*)

Rotate Z array by angle alpha.

Rotation angle must be given in degrees. All angles are referenced to geographic North, positive in clockwise direction. (Mathematically negative!)

In non-rotated state, X refs to North and Y to East direction.

**Updates the attributes**

- *z*
- *z_err*
- *zrot*
- *resistivity*
- *phase*
- *resistivity_err*
- *phase_err*

**skew**

Returns the skew of Z as defined by Z[0, 1] + Z[1, 0]

**Note:** This is not the MT skew, but simply the linear algebra skew

**Returns** skew

**Return type** np.ndarray(nfreq, 2, 2)

**skew_err**

Returns the skew error of Z as defined by Z_err[0, 1] + Z_err[1, 0]

**Note:** This is not the MT skew, but simply the linear algebra skew

**Returns** skew_err

**Return type** np.ndarray(nfreq, 2, 2)

**trace**
Return the trace of Z

**Returns** Trace(z)

**Return type** np.ndarray(nfreq, 2, 2)

**trace_err**
Return the trace of Z

**Returns** Trace(z)

**Return type** np.ndarray(nfreq, 2, 2)

**z**
Impedance tensor

np.ndarray(nfreq, 2, 2)

mtpy.core.z.**correct4sensor_orientation**(*Z_prime*, *Bx=0*, *By=90*, *Ex=0*, *Ey=90*, *Z_prime_error=None*)
Correct a Z-array for wrong orientation of the sensors.

**Assume, E' is measured by sensors orientated with the angles** E'x: a E'y: b

**Assume, B' is measured by sensors orientated with the angles** B'x: c B'y: d

**With those data, one obtained the impedance tensor Z':** E' = Z' * B'

**Now we define change-of-basis matrices T,U so that** E = T * E' B = U * B'

=> T contains the expression of the E'-basis in terms of E (the standard basis) and U contains the expression of the B'-basis in terms of B (the standard basis) The respective expressions for E'x-basis vector and E'y-basis vector are the columns of T. The respective expressions for B'x-basis vector and B'y-basis vector are the columns of U.

We obtain the impedance tensor in default coordinates as:

**E' = Z' * B' => T^(-1) * E = Z' * U^(-1) * B** => E = T * Z' * U^(-1) * B => Z = T * Z' * U^(-1)

**Parameters**

- **Z_prime** – impedance tensor to be adjusted
- **Bx** (*float (angle in degrees)*) – orientation of Bx relative to geographic north (0) *default* is 0
- **By** –
- **Ex** (*float (angle in degrees)*) – orientation of Ex relative to geographic north (0) *default* is 0
- **Ey** (*float (angle in degrees)*) – orientation of Ey relative to geographic north (0) *default* is 90
- **Z_prime_error** (*np.ndarray(Z_prime.shape)*) – impedance tensor error (std) *default* is None

**Dtype Z_prime** np.ndarray(num_freq, 2, 2, dtype='complex')

**Returns** adjusted impedance tensor

**Return type** np.ndarray(Z_prime.shape, dtype='complex')

**Returns** impedance tensor standard deviation in default orientation

**Return type** np.ndarray(Z_prime.shape, dtype='real')

## 1.2 Module TS

## 1.3 Module MT

## 1.4 Module EDI

## 1.5 Module XML

**Note:** This module is written to align with the tools written by Anna Kelbert <akelbert@usgs.gov>

**class** mtpy.core.mt_xml.**MT_XML**(***kwargs*)

Class to read and write MT information from XML format. This tries to follow the format put forward by Anna Kelbert for archiving MT response data.

A configuration file can be read in that might make it easier to write multiple files for the same survey.

**See also:**

mtpy.core.mt_xml.XML_Config

| Attributes | Description |
|---|---|
| Z | object of type mtpy.core.z.Z |
| Tipper | object of type mtpy.core.z.Tipper |

**Note:** All other attributes are of the same name and of type XML_element, where attributes are name, value and attr. Attr contains any tag information. This is left this way so that mtpy.core.mt.MT can read in the information. **Use mtpy.core.mt.MT for conversion between data formats.**

| Methods | Description |
|---|---|
| read_cfg_file | Read a configuration file in the format of XML_Config |
| read_xml_file | Read an xml file |
| write_xml_file | Write an xml file |

**Example** :: >>> import mtpy.core.mt_xml as mtxml >>> x = mtxml.read_xml_file(r"/home/mt_data/mt01.xml") >>> x.read_cfg_file(r"/home/mt_data/survey_xml.cfg") >>> x.write_xml_file(r"/home/mt_data/xml/mt01.xml")

**Tipper**
get Tipper information

**Z**
get z information

**read_xml_file**(*xml_fn*)
read in an xml file and set attributes appropriately.

**xml_fn** [string] full path of xml file to read in

**write_xml_file**(*xml_fn*, *cfg_fn=None*)
write xml from edi

**exception** mtpy.core.mt_xml.**MT_XML_Error**

**class** mtpy.core.mt_xml.**XML_Config**(***kwargs*)

Class to deal with configuration files for xml.

Includes all the important information for the station and how data was processed.

Key Information includes:

| Name | Purpose |
|---|---|
| ProductID | Station name |
| ExternalUrl | External URL to link to data |
| Notes | Any important information on station, data collection. |
| TimeSeriesArchived | Information on Archiving time series including URL. |
| Image | A location to an image of the station or the MT response. |

- **ProductID –> station name**
  - ExternalUrl –> external url to link to data
  - Notes –> any

**read_cfg_file**(*cfg_fn=None*)

Read in a cfg file making all key = value pairs attribures of XML_Config. Being sure all new attributes are XML_element objects.

**The assumed structure of the xml.cfg file is similar to:** ‘‘# XML Configuration File MTpy

Attachement.Description = Original file use to produce XML Attachment.Filename = None

Copyright.Citation.Authors = None Copyright.Citation.DOI = None Copyright.Citation.Journal = None Copyright.Citation.Title = None Copyright.Citation.Volume = None Copyright.Citation.Year = None

PeriodRange(max=0)(min=0) = None‘‘

where the heirarchy of information is separated by a . and if the information has attribures they are in the name with (key=value) syntax.

**cfg_fn** [string] full path to cfg file to read in

**Read in xml.cfg file**

```
>>> import mtpy.core.mtxml as mtxml
>>> cfg_obj = mtxml.XML_Config()
>>> cfg_obj.read_cfg_file(r"/home/MT/xml.cfg")
```

**write_cfg_file**(*cfg_fn=None*)

Write out configuration file in the style of: parent.attribute = value

**cfg_fn** [string] full path to write the configuration file

**class** mtpy.core.mt_xml.**XML_element**(*name*, *attr*, *value*, ***kwargs*)

**Basically an ET element. The key components are**

- ‘name’ –> name of the element
- ‘attr’ –> attribute information of the element
- ‘value’ –> value of the element

Used the property function here to be sure that these 3 cannot be set through the common k.value = 10, just in case there are similar names in the xml file. This seemed to be the safest to avoid those cases.

## 1.6 Module JFile

**class** mtpy.core.jfile.**JFile**(*j_fn=None*)

be able to read and write a j-file

**read_header**(*j_lines=None*)

Parsing the header lines of a j-file to extract processing information.

Input: - j-file as list of lines (output of readlines())

Output: - Dictionary with all parameters found

**read_j_file**(*j_fn=None*)

read_j_file will read in a *.j file output by BIRRP (better than reading lots of *.<k>r<l>.rf files)

Input: j-filename

Output: 4-tuple - periods : N-array - Z_array : 2-tuple - values and errors - tipper_array : 2-tuple - values and errors - processing_dict : parsed processing parameters from j-file header

**read_metadata**(*j_lines=None*, *j_fn=None*)

read in the metadata of the station, or information of station logistics like: lat, lon, elevation

Not really needed for a birrp output since all values are nan's

# CHAPTER 2

# Analysis

## 2.1 Module Distortion

## 2.2 Module Geometry

## 2.3 Module Phase Tensor

## 2.4 Module Static Shift

## 2.5 Module Z Invariants

Created on Wed May 08 09:40:42 2013

Interpreted from matlab code written by Stephan Thiel 2005

@author: jpeacock

**class** `mtpy.analysis.zinvariants.`**`Zinvariants`**(*z_object=None*, *z_array=None*, *z_err_array=None*, *freq=None*, *rot_z=0*)

calculates invariants from Weaver et al. [2000, 2003]. At the moment it does not calculate the error for each invariant, only the strike.

**z_object** [type mtpy.core.z]

**needs to have attributes:** *z –> np.array((nf, 2, 2), dtype='complex')

*z_err –> np.array((nf, 2, 2), dtype='real')

*freq –> np.array(nf)

**z** [complex np.array(nf,2,2)] impedance tensor array

**z_err** [real np.array(nf,2,2)] impedance tensor error array

**freq** [np.array(nf)] array of freq cooresponding to the impedance tensor elements.

**inv1** : real off diaganol part normalizing factor

**inv2** : imaginary off diaganol normalizing factor

**inv3** : real anisotropy factor (range from [0,1])

**inv4** : imaginary anisotropy factor (range from [0,1])

**inv5** : suggests electric field twist

**inv6** : suggests in phase small scale distortion

**inv7** : suggests 3D structure

**strike** : strike angle (deg) assuming positive clockwise 0=N

**strike_err** : strike angle error (deg)

**q** : dependent variable suggesting dimensionality

**Weaver, J. T., Agarwal, A. K., Lilley, F. E. M., 2000,** Characterization of the magnetotelluric tensor in terms of its invariants, Geophysical Journal International, 141, 321–336.

**Weaver, J. T., Agarwal, A. K., Lilley, F. E. M., 2003,** The relationship between the magnetotelluric tensor invariants and the phase tensor of Caldwell, Bibby and Brown, presented at 3D Electromagnetics III, ASEG, paper 43.

**Lilley, F. E. M, 1998, Magnetotelluric tensor dcomposition: 1: Theory** for a basic procedure, Geophysics, 63, 1885–1897.

**Lilley, F. E. M, 1998, Magnetotelluric tensor dcomposition: 2: Examples** of a basic procedure, Geophysics, 63, 1898–1907.

**Szarka, L. and Menvielle, M., 1997, Analysis of rotational invariants** of the magnetotelluric impedance tensor, Geophysical Journal International, 129, 133–142.

**compute_invariants**()

Computes the invariants according to Weaver et al., [2000, 2003]

Mostly used to plot Mohr's circles

In a 1D case: rho = mu (inv1**2+inv2**2)/w & phi = arctan(inv2/inv1)

**Sets the invariants as attributes:** **inv1** : real off diaganol part normalizing factor

**inv2** : imaginary off diaganol normalizing factor

**inv3** : real anisotropy factor (range from [0,1])

**inv4** : imaginary anisotropy factor (range from [0,1])

**inv5** : suggests electric field twist

**inv6** : suggests in phase small scale distortion

**inv7** : suggests 3D structure

**strike** : strike angle (deg) assuming positive clockwise 0=N

**strike_err** : strike angle error (deg)

**q** : dependent variable suggesting dimensionality

**rotate**(*rot_z*)

Rotates the impedance tensor by the angle rot_z clockwise positive assuming 0 is North

**set_freq**(*freq*)
set the freq array, needs to be the same length at z

**set_z**(*z_array*)
set the z array.

If the shape changes or the freq are changed need to input those as well.

**set_z_err**(*z_err_array*)
set the z_err array.

If the shape changes or the freq are changed need to input those as well.

# CHAPTER 3

# Modeling

## 3.1 Module ModEM

## 3.2 Module Occam 1D

## 3.3 Module Occam 2D

## 3.4 Module Winglink

Created on Mon Aug 22 15:19:30 2011

deal with output files from winglink.

@author: jp

**class** `mtpy.modeling.winglink.`**`PlotMisfitPseudoSection`**(*data_fn*, *resp_fn*, ***kwargs*)

plot a pseudo section misfit of the data and response if given

**Note:** the output file from winglink does not contain errors, so to get a normalized error, you need to input the error for each component as a percent for resistivity and a value for phase and tipper. If you used the data errors, unfortunately, you have to input those as arrays.

**wl_data_fn** [string] full path to output data file from winglink

| key words | description |
|---|---|
| axmpte | matplotlib.axes instance for TE model phase |
| axmptm | matplotlib.axes instance for TM model phase |

Continued on next page

Table 1 – continued from previous page

| key words | description |
|---|---|
| axmrte | matplotlib.axes instance for TE model app. res |
| axmrtm | matplotlib.axes instance for TM model app. res |
| axpte | matplotlib.axes instance for TE data phase |
| axptm | matplotlib.axes instance for TM data phase |
| axrte | matplotlib.axes instance for TE data app. res. |
| axrtm | matplotlib.axes instance for TM data app. res. |
| cb_pad | padding between colorbar and axes |
| cb_shrink | percentage to shrink the colorbar to |
| fig | matplotlib.figure instance |
| fig_dpi | resolution of figure in dots per inch |
| fig_num | number of figure instance |
| fig_size | size of figure in inches (width, height) |
| font_size | size of font in points |
| label_list | list to label plots |
| ml | factor to label stations if 2 every other station is labeled on the x-axis |
| period | np.array of periods to plot |
| phase_cmap | color map name of phase |
| phase_limits_te | limits for te phase in degrees (min, max) |
| phase_limits_tm | limits for tm phase in degrees (min, max) |
| plot_resp | [ 'y' \| 'n' ] to plot response |
| plot_yn | [ 'y' \| 'n' ] 'y' to plot on instantiation |
| res_cmap | color map name for resistivity |
| res_limits_te | limits for te resistivity in log scale (min, max) |
| res_limits_tm | limits for tm resistivity in log scale (min, max) |
| rp_list | list of dictionaries as made from read2Dresp |
| station_id | index to get station name (min, max) |
| station_list | station list got from rp_list |
| subplot_bottom | subplot spacing from bottom (relative coordinates) |
| subplot_hspace | vertical spacing between subplots |
| subplot_left | subplot spacing from left |
| subplot_right | subplot spacing from right |
| subplot_top | subplot spacing from top |
| subplot_wspace | horizontal spacing between subplots |

| Methods | Description |
|---|---|
| plot | plots a pseudo-section of apparent resistiviy and phase of data and model if given. called on instantiation if plot_yn is 'y'. |
| redraw_plot | call redraw_plot to redraw the figures, if one of the attributes has been changed |
| save_figure | saves the matplotlib.figure instance to desired location and format |

**Example**

```
>>> import mtpy.modeling.occam2d as occam2d
>>> ocd = occam2d.Occam2DData()
>>> rfile = r"/home/Occam2D/Line1/Inv1/Test_15.resp"
>>> ocd.data_fn = r"/home/Occam2D/Line1/Inv1/DataRW.dat"
>>> ps1 = ocd.plot2PseudoSection(resp_fn=rfile)
```

**get_misfit**()
compute misfit of MT response found from the model and the data.

Need to normalize correctly

**plot**()
plot pseudo section of data and response if given

**redraw_plot**()
redraw plot if parameters were changed

use this function if you updated some attributes and want to re-plot.

**Example**

```
>>> # change the color and marker of the xy components
>>> import mtpy.modeling.occam2d as occam2d
>>> ocd = occam2d.Occam2DData(r"/home/occam2d/Data.dat")
>>> p1 = ocd.plotPseudoSection()
>>> #change color of te markers to a gray-blue
>>> p1.res_cmap = 'seismic_r'
>>> p1.redraw_plot()
```

**save_figure**(*save_fn*, *file_format='pdf'*, *orientation='portrait'*, *fig_dpi=None*, *close_plot='y'*)
save_plot will save the figure to save_fn.

**save_fn** [string] full path to save figure to, can be input as * directory path -> the directory path to save to

in which the file will be saved as save_fn/station_name_PhaseTensor.file_format

- full path -> file will be save to the given path. If you use this option then the format will be assumed to be provided by the path

**file_format** [[ pdf | eps | jpg | png | svg ]] file type of saved figure pdf,svg,eps...

**orientation** [[ landscape | portrait ]] orientation in which the file will be saved *default* is portrait

**fig_dpi** [int] The resolution in dots-per-inch the file will be saved. If None then the dpi will be that at which the figure was made. I don't think that it can be larger than dpi of the figure.

**close_plot** [[ y | n ]]

- 'y' will close the plot after saving.
- 'n' will leave plot open

**Example**

```
>>> # to save plot as jpg
>>> import mtpy.modeling.occam2d as occam2d
>>> dfn = r"/home/occam2d/Inv1/data.dat"
>>> ocd = occam2d.Occam2DData(dfn)
>>> ps1 = ocd.plotPseudoSection()
>>> ps1.save_plot(r'/home/MT/figures', file_format='jpg')
```

**update_plot**()
update any parameters that where changed using the built-in draw from canvas.

Use this if you change an of the .fig or axes properties

**Example**

```
>>> # to change the grid lines to only be on the major ticks
>>> import mtpy.modeling.occam2d as occam2d
>>> dfn = r"/home/occam2d/Inv1/data.dat"
>>> ocd = occam2d.Occam2DData(dfn)
>>> ps1 = ocd.plotPseudoSection()
>>> [ax.grid(True, which='major') for ax in [ps1.axrte,ps1.axtep]]
>>> ps1.update_plot()
```

**class** mtpy.modeling.winglink.**PlotPseudoSection**(*wl_data_fn=None*, ***kwargs*)

plot a pseudo section of the data and response if given

**wl_data_fn** [string] full path to winglink output data file.

| key words | description |
|---|---|
| axmpte | matplotlib.axes instance for TE model phase |
| axmptm | matplotlib.axes instance for TM model phase |
| axmrte | matplotlib.axes instance for TE model app. res |
| axmrtm | matplotlib.axes instance for TM model app. res |
| axpte | matplotlib.axes instance for TE data phase |
| axptm | matplotlib.axes instance for TM data phase |
| axrte | matplotlib.axes instance for TE data app. res. |
| axrtm | matplotlib.axes instance for TM data app. res. |
| cb_pad | padding between colorbar and axes |
| cb_shrink | percentage to shrink the colorbar to |
| fig | matplotlib.figure instance |
| fig_dpi | resolution of figure in dots per inch |
| fig_num | number of figure instance |
| fig_size | size of figure in inches (width, height) |
| font_size | size of font in points |
| label_list | list to label plots |
| ml | factor to label stations if 2 every other station is labeled on the x-axis |
| period | np.array of periods to plot |
| phase_cmap | color map name of phase |
| phase_limits_te | limits for te phase in degrees (min, max) |
| phase_limits_tm | limits for tm phase in degrees (min, max) |
| plot_resp | [ 'y' \| 'n' ] to plot response |
| plot_tipper | [ 'y' \| 'n' ] to plot tipper |
| plot_yn | [ 'y' \| 'n' ] 'y' to plot on instantiation |
| res_cmap | color map name for resistivity |
| res_limits_te | limits for te resistivity in log scale (min, max) |
| res_limits_tm | limits for tm resistivity in log scale (min, max) |
| rp_list | list of dictionaries as made from read2Dresp |
| station_id | index to get station name (min, max) |
| station_list | station list got from rp_list |
| subplot_bottom | subplot spacing from bottom (relative coordinates) |
| subplot_hspace | vertical spacing between subplots |
| subplot_left | subplot spacing from left |
| subplot_right | subplot spacing from right |
| subplot_top | subplot spacing from top |
| subplot_wspace | horizontal spacing between subplots |

| Methods | Description |
|---|---|
| plot | plots a pseudo-section of apparent resistiviy and phase of data and model if given. called on instantiation if plot_yn is 'y'. |
| redraw_plot | call redraw_plot to redraw the figures, if one of the attributes has been changed |
| save_figure | saves the matplotlib.figure instance to desired location and format |

**Example**

```
>>> import mtpy.modeling.winglink as winglink
>>> d_fn = r"/home/winglink/Line1/Inv1/DataRW.txt"
>>> ps_plot = winglink.PlotPseudoSection(d_fn)
```

**plot**()
plot pseudo section of data and response if given

**redraw_plot**()
redraw plot if parameters were changed

use this function if you updated some attributes and want to re-plot.

**Example**

```
>>> # plot tipper and change station id
>>> import mtpy.modeling.winglink as winglink
>>> ps_plot = winglink.PlotPseudosection(wl_fn)
>>> ps_plot.plot_tipper = 'y'
>>> ps_plot.station_id = [2, 5]
>>> #label only every 3rd station
>>> ps_plot.ml = 3
>>> ps_plot.redraw_plot()
```

**save_figure**(*save_fn*, *file_format='pdf'*, *orientation='portrait'*, *fig_dpi=None*, *close_plot='y'*)
save_plot will save the figure to save_fn.

**save_fn** [string] full path to save figure to, can be input as * directory path -> the directory path to save to

in which the file will be saved as save_fn/station_name_PhaseTensor.file_format

- full path -> file will be save to the given path. If you use this option then the format will be assumed to be provided by the path

**file_format** [[ pdf | eps | jpg | png | svg ]] file type of saved figure pdf,svg,eps...

**orientation** [[ landscape | portrait ]] orientation in which the file will be saved *default* is portrait

**fig_dpi** [int] The resolution in dots-per-inch the file will be saved. If None then the dpi will be that at which the figure was made. I don't think that it can be larger than dpi of the figure.

**close_plot** [[ y | n ]]

- 'y' will close the plot after saving.
- 'n' will leave plot open

**Example**

```
>>> # to save plot as jpg
>>> ps_plot.save_plot(r'/home/MT/figures', file_format='jpg')
```

**update_plot**()

update any parameters that where changed using the built-in draw from canvas.

Use this if you change an of the .fig or axes properties

**Example**

```
>>> # to change the grid lines to only be on the major ticks
>>> [ax.grid(True, which='major') for ax in [ps_plot.axrte]]
>>> ps_plot.update_plot()
```

**class** mtpy.modeling.winglink.**PlotResponse**(*wl_data_fn=None*, *resp_fn=None*, ***kwargs*)

Helper class to deal with plotting the MT response and occam2d model.

**data_fn** [string] full path to data file

**resp_fn** [string or list] full path(s) to response file(s)

| Attributes/key words | description |
|---|---|
| ax_list | list of matplotlib.axes instances for use with OccamPointPicker |
| color_mode | [ 'color' \| 'bw' ] plot figures in color or black and white ('bw') |
| cted | color of Data TE marker and line |
| ctem | color of Model TE marker and line |
| ctewl | color of Winglink Model TE marker and line |
| ctmd | color of Data TM marker and line |
| ctmm | color of Model TM marker and line |
| ctmwl | color of Winglink Model TM marker and line |
| e_capsize | size of error bar caps in points |
| e_capthick | line thickness of error bar caps in points |
| err_list | list of line properties of error bars for use with OccamPointPicker |
| fig_dpi | figure resolution in dots-per-inch |
| fig_list | list of dictionaries with key words station –> station name fig –> matplotlib.figure instance axrte –> matplotl |
| fig_num | starting number of figure |
| fig_size | size of figure in inches (width, height) |
| font_size | size of axes ticklabel font in points |
| line_list | list of matplotlib.Line instances for use with OccamPointPicker |
| lw | line width of lines in points |
| ms | marker size in points |
| mted | marker for Data TE mode |
| mtem | marker for Model TE mode |
| mtewl | marker for Winglink Model TE |
| mtmd | marker for Data TM mode |
| mtmm | marker for Model TM mode |
| mtmwl | marker for Winglink TM mode |
| period | np.ndarray of periods to plot |
| phase_limits | limits on phase plots in degrees (min, max) |
| plot_model_error | [ 'y' \| 'n' ] *default* is 'y' to plot model errors |
| plot_num | [ 1 \| 2 ] 1 to plot both modes in a single plot 2 to plot modes in separate plots (default) |
| plot_tipper | [ 'y' \| 'n' ] plot tipper data if desired |
| plot_type | [ '1' \| station_list] '1' –> to plot all stations in different figures station_list –> to plot a few stations, give nam |

| Attributes/key words | description |
|---|---|
| plot_yn | [ 'y' \| 'n'] 'y' –> to plot on instantiation 'n' –> to not plot on instantiation |
| res_limits | limits on resistivity plot in log scale (min, max) |
| rp_list | list of dictionaries from read2Ddata |
| station_list | station_list list of stations in rp_list |
| subplot_bottom | subplot spacing from bottom (relative coordinates) |
| subplot_hspace | vertical spacing between subplots |
| subplot_left | subplot spacing from left |
| subplot_right | subplot spacing from right |
| subplot_top | subplot spacing from top |
| subplot_wspace | horizontal spacing between subplots |
| wl_fn | Winglink file name (full path) |

| Methods | Description |
|---|---|
| plot | plots the apparent resistiviy and phase of data and model if given. called on instantiation if plot_yn is 'y'. |
| re-draw_plot | call redraw_plot to redraw the figures, if one of the attributes has been changed |
| save_figures | save all the matplotlib.figure instances in fig_list |

**Example** :: >>> data_fn = r"/home/occam/line1/inv1/OccamDataFile.dat" >>> resp_list = [r"/home/occam/line1/inv1/test_{0:02}".format(ii)

for ii in range(2, 8, 2)]

```
>>> pr_obj = occam2d.PlotResponse(data_fn, resp_list, plot_tipper='y')
```

**plot** ()

plot the data and model response, if given, in individual plots.

**redraw_plot** ()

redraw plot if parameters were changed

use this function if you updated some attributes and want to re-plot.

**Example**

```
>>> # change the color and marker of the xy components
>>> import mtpy.modeling.occam2d as occam2d
>>> ocd = occam2d.Occam2DData(r"/home/occam2d/Data.dat")
>>> p1 = ocd.plot2DResponses()
>>> #change color of te markers to a gray-blue
>>> p1.cted = (.5, .5, .7)
>>> p1.redraw_plot()
```

**save_figures** (*save_path*, *fig_fmt='pdf'*, *fig_dpi=None*, *close_fig='y'*)

save all the figure that are in self.fig_list

**Example**

```
>>> # change the color and marker of the xy components
>>> import mtpy.modeling.occam2d as occam2d
>>> ocd = occam2d.Occam2DData(r"/home/occam2d/Data.dat")
>>> p1 = ocd.plot2DResponses()
>>> p1.save_figures(r"/home/occam2d/Figures", fig_fmt='jpg')
```

**exception** mtpy.modeling.winglink.**WLInputError**

mtpy.modeling.winglink.**read_model_file**(*model_fn*)

readModelFile reads in the XYZ txt file output by Winglink.

**Inputs:** modelfile = fullpath and filename to modelfile profiledirection = 'ew' for east-west predominantly, 'ns' for

predominantly north-south. This gives column to fix

mtpy.modeling.winglink.**read_output_file**(*output_fn*)

Reads in an output file from winglink and returns the data in the form of a dictionary of structured arrays.

**output_fn** [string] the full path to winglink outputfile

**wl_data** [dictionary with keys of station names] each station contains a structured array with keys * 'station' –> station name * 'period' –> periods to plot * 'te_res' –> TE resistivity in linear scale * 'tm_res' –> TM resistivity in linear scale * 'te_phase' –> TE phase in deg * 'tm_phase' –> TM phase in deg * 're_tip' –> real tipper amplitude. * 'im_tip' –> imaginary tipper amplitude * 'rms' –> RMS for the station * 'index' –> order from left to right of station number

**Note:** each data is an np.ndarray(2, num_periods) where the first index is the data and the second index is the model response

## 3.5 Module WS3DINV

# CHAPTER 4

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## j



## m



## z

# Index

## C



## D



## F



## G



## I



## J



## M



## N



## O



## P



## R

## S



## T



## U



## W



## X



## Z