
MTpy-v2 Documentation

Release 2.0.12

Jared Peacock

Dec 04, 2024

GENERAL INFORMATION

1 Examples	3
2 Indices and tables	421
Python Module Index	423

mtpy provides tools for working with magnetotelluric (MT) data. MTpy-v2 is an update version of [mtpy](<https://github.com/MTgeophysics/mtpy>). Many things have changed under the hood and usage is different from mtpy v1. The main difference is that there is a central data type that can hold transfer functions and then read/write to your modeling program, plot, and analyze your data. No longer will you need a directory of EDI files and then read them in everytime you want to do something. You only need to build a project once and save it to an MTH5 file and you are ready to go. All metadata uses [mt-metadata](<https://github.com/kujaku11/mt-metadata>).

Because the workflow has changed from mtpy v1, there are example notebooks to demonstrate the new workflow see *Examples*.

EXAMPLES

Click on the *Binder* badge above to interact with Jupyter Notebook examples. There are example notebooks in

- [docs/source/examples/notebooks](#)

1.1 Installation

1.1.1 Stable release

PIP

To install *mt_metadata*, run this command in your terminal:

```
$ pip install mtpy-v2
```

This is the preferred method to install *mt_metadata*, as it will always install the most recent stable release.

If you don't have [pip](#) installed, this Python installation guide can guide you through the process.

Conda-Forge

To install *mtpy-v2*, run either of these commands in your Conda terminal (<https://conda-forge.org/#about>):

```
$ conda install -c conda-forge mtpy-v2
```

or

```
$ conda config --add channels conda-forge
$ conda config --set channel_priority strict
$ conda install mtpy-v2
```

Note

If you are updating *mt_metadata* you should use the same installer as your previous version or remove the current version and do a fresh install.

1.1.2 From sources

The sources for MTH5 can be downloaded from the [Github](#) repo.

You can either clone the public repository:

```
$ git clone https://github.com/MTgeophysics/mtpy-v2
```

Or download the [tarball](#):

```
$ curl -OJL https://github.com/MTgeophysics/mtpy-v2/tarball/main
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

1.2 Usage

1.2.1 MT Object

MT transfer functions come in all kinds of formats and flavors. The goal of MT is to centralize and standardize an MT transfer function with common metadata and accessibility to the data. MT inherits `mt_metadata.transfer_function.core.TF <https://mt-metadata.readthedocs.io/en/latest/source/tf_structure.html>`__ which has the ability to read/write in various file types. If there is a file type that is not supported yet raise an issue in [mt-metadata](#).

Format	Description	Extension	Read	Write
EDI	Common SEG format	.edi	yes	yes
EMTF XML	Anna Kelbert's XML Format for archiving at IRIS	.xml	yes	yes
Z-Files	Output from Gary Egberts processing code	.zmm, .zss, .zrr	yes	yes
J-Files	Alan Jones' format and output of Alan Chave's BIRRP code	.j	yes	no
Zonge AVG	Zonge International processing code output	.avg	yes	no

The MT has a couple of important attributes and method that are described below as we progress through an example file. Here we will look at an EMTF XML because this format has the most comprehensive metadata so far.

```
[1]: from mtpy import MT
      from mt_metadata import TF_XML
```

```
[2]: mt_object = MT(TF_XML)
      mt_object.read()
```

TF Metadata

Important in describing the transfer function are metadata attributes, namely the location, what survey the station was collected in, timing, and how the data were processed. These are contained in logical metadata objects. For further reading on metadata objects see [MT-metadata](#)

- `MT.survey_metadata`: describes the general survey details that this transfer function belongs to.
- `MT.station_metadata`: describes the station location, timing, runs processed, processing scheme.
 - `MT.station_metadata.transfer_function`: describes how the data were processed.
 - `MT.station_metadata.runs`: provides details on the runs processed, timing, sample rate, channels recorded, data logger details.
 - * `MT.station_metadata.runs[run_id].channels`: describes channel metadata including timing, sensors, location.

Survey Metadata

Survey metadata provides information about the survey ID, geographic locations, who acquired the data, is there a DOI associated with the data or publications, how the data can be used, licenses, and general information about the overall survey.

```
[3]: mt_object.survey_metadata
[3]: {
    "survey": {
        "acquired_by.author": "National Geoelectromagnetic Facility",
        "citation_dataset.authors": "Schultz, A., Pellerin, L., Bedrosian, P., Kelbert, A., Crosbie, J.",
        "citation_dataset.doi": "doi:10.17611/DP/EMTF/USMTARRAY/SOUTH",
        "citation_dataset.title": "USMTArray South Magnetotelluric Transfer Functions",
        "citation_dataset.year": "2020-2023",
        "citation_journal.doi": null,
        "comments": "copyright.acknowledgement:The USMTArray-CONUS South campaign was carried out through a cooperative agreement between\nthe U.S. Geological Survey (USGS) and Oregon State University (OSU). A subset of 40 stations\nin the SW US were funded through NASA grant 80NSSC19K0232.\nLand permitting, data acquisition, quality control and field processing were\nncarried out by Green Geophysics with project management and instrument/engineering\nsupport from OSU and Chaytus Engineering, respectively.\nProgram oversight, definitive data processing and data archiving were provided\nby the USGS Geomagnetism Program and the Geology, Geophysics and Geochemistry Science Centers.\nWe thank the U.S. Forest Service, the Bureau of Land Management, the National Park Service,\nthe Department of Defense, numerous state land offices and the many private landowners\nwho permitted land access to acquire the USMTArray data.;\ncopyright.conditions_of_use:All data and metadata for this survey are available free\nof charge and may be copied freely, duplicated and further distributed provided that\nthis data set is cited as the reference, and that the author(s) contributions are\nacknowledged as detailed in the Acknowledgements. Any papers cited in this file are\nonly for reference. There is no requirement to cite these papers when the data are\nused. Whenever possible, we ask that the author(s) are notified prior to any\npublication that makes use of these data.\nWhile the author(s) strive to provide data\nand metadata of best possible quality, neither the author(s) of this data set, nor\nIRIS make any claims, promises, or guarantees about the accuracy, completeness, or\nadequacy of this information, and expressly disclaim liability for errors and\nomissions in the contents of this file. Guidelines about the quality or limitations of\nthe data and metadata, as obtained from the author(s), are included for informational\npurposes only.; copyright.release_status:Unrestricted Release",
        "country": [
            "USA"
        ],
        "datum": "WGS84",
        "geographic_name": "CONUS South",
        "id": "CONUS South",
        "name": null,
        "northwest_corner.latitude": 34.470528,
        "northwest_corner.longitude": -108.712288,
        "project": "USMTArray",
        "project_lead.email": null,
        "project_lead.organization": null,
        "release_license": "CC0-1.0",
    }
}
```

(continues on next page)

(continued from previous page)

```

    "southeast_corner.latitude": 34.470528,
    "southeast_corner.longitude": -108.712288,
    "summary": "Magnetotelluric Transfer Functions",
    "time_period.end_date": "2020-10-07",
    "time_period.start_date": "2020-09-20"
}
}

```

Station Metadata

Station metadata is the most important to describe the transfer function, it provides ID, location, timing and then specifics on how the data were processed, run metadata, and channel metadata.

[4]: `mt_object.station_metadata`

```

[4]: {
    "station": {
        "acquired_by.author": "National Geoelectromagnetic Facility",
        "channels_recorded": [
            "ex",
            "ey",
            "hx",
            "hy",
            "hz"
        ],
        "comments": "description:Magnetotelluric Transfer Functions; primary_data.  

        ↪filename:NMX20b_NMX20_NMWF20_COR21_NMY21-NMX20b_NMX20_UTS18.png; attachment.description:  

        ↪The original used to produce the XML; attachment.filename:NMX20b_NMX20_NMWF20_COR21_  

        ↪NMY21-NMX20b_NMX20_UTS18.zmm; site.data_quality_notes.comments.author:Jade Crosbie,  

        ↪Paul Bedrosian and Anna Kelbert; site.data_quality_notes.comments.value:great TF from  

        ↪10 to 10000 secs (or longer)",
        "data_type": "mt",
        "fdsn.id": "USMTArray.NMX20.2020",
        "geographic_name": "Nations Draw, NM, USA",
        "id": "NMX20",
        "location.datum": "WGS84",
        "location.declination.epoch": "2020.0",
        "location.declination.model": "WMM",
        "location.declination.value": 9.09,
        "location.elevation": 1940.05,
        "location.latitude": 34.470528,
        "location.longitude": -108.712288,
        "orientation.angle_to_geographic_north": 0.0,
        "orientation.method": null,
        "orientation.reference_frame": "geographic",
        "provenance.archive.comments": "IRIS DMC MetaData",
        "provenance.archive.name": null,
        "provenance.archive.url": "http://www.iris.edu/mda/ZU/NMX20",
        "provenance.creation_time": "2021-03-17T14:47:44+00:00",
        "provenance.creator.author": "Jade Crosbie, Paul Bedrosian and Anna Kelbert",
        "provenance.creator.email": "pbedrosian@usgs.gov",
        "provenance.creator.name": "Jade Crosbie, Paul Bedrosian and Anna Kelbert",
    }
}

```

(continues on next page)

(continued from previous page)

```

"provenance.creator.organization": "U.S. Geological Survey",
"provenance.creator.url": "https://www.usgs.gov/natural-hazards/geomagnetism",
"provenance.software.author": null,
"provenance.software.name": "EMTF File Conversion Utilities 4.0",
"provenance.software.version": null,
"provenance.submitter.author": "Anna Kelbert",
"provenance.submitter.email": "akelbert@usgs.gov",
"provenance.submitter.name": "Anna Kelbert",
"provenance.submitter.organization": "U.S. Geological Survey, Geomagnetism
Program",
"provenance.submitter.url": "https://www.usgs.gov/natural-hazards/geomagnetism",
"release_license": "CC0-1.0",
"run_list": [
    "NMX20a",
    "NMX20b"
],
"time_period.end": "2020-10-07T20:28:00+00:00",
"time_period.start": "2020-09-20T19:03:06+00:00",
"transfer_function.coordinate_system": "geographic",
"transfer_function.data_quality.good_from_period": 5.0,
"transfer_function.data_quality.good_to_period": 29127.0,
"transfer_function.data_quality.rating.value": 5,
"transfer_function.id": "NMX20",
"transfer_function.processed_by.author": "Jade Crosbie, Paul Bedrosian and Anna
Kelbert",
"transfer_function.processed_by.name": "Jade Crosbie, Paul Bedrosian and Anna
Kelbert",
"transfer_function.processed_date": "1980-01-01",
"transfer_function.processing_parameters": [],
"transfer_function.processing_type": "Robust Multi-Station Reference",
"transfer_function.remote_references": [
    "NMW20",
    "COR21",
    "UTS18"
],
"transfer_function.runs_processed": [
    "NMX20a",
    "NMX20b"
],
"transfer_function.sign_convention": "exp(+ i\omega t)",
"transfer_function.software.author": "Gary Egbert",
"transfer_function.software.last_updated": "2015-08-26",
"transfer_function.software.name": "EMTF",
"transfer_function.software.version": null,
"transfer_function.units": null
}
}

```

Run Metadata

Run metadata is located in `MT.station_metadata.runs` which is a list-dictionary object that contains the runs used for processing.

```
[5]: mt_object.station_metadata.runs

[5]: OrderedDict([('NMX20a', {
    "run": {
        "channels_recorded_auxiliary": [],
        "channels_recorded_electric": [
            "ex",
            "ey"
        ],
        "channels_recorded_magnetic": [
            "hx",
            "hy",
            "hz"
        ],
        "comments": "comments.author:Isaac Sageman; comments.value:X array at 0 deg_\nrotation. All e-lines 50m. Soft sandy dirt. Water tank ~400m NE. County Rd 601 ~200m_\nSE. Warm sunny day.",
        "data_logger.firmware.author": null,
        "data_logger.firmware.name": null,
        "data_logger.firmware.version": null,
        "data_logger.id": "2612-01",
        "data_logger.manufacturer": "Barry Narod",
        "data_logger.timing_system.drift": 0.0,
        "data_logger.timing_system.type": "GPS",
        "data_logger.timing_system.uncertainty": 0.0,
        "data_logger.type": "NIMS",
        "data_type": "BBMT",
        "id": "NMX20a",
        "sample_rate": 1.0,
        "time_period.end": "2020-09-20T19:29:28+00:00",
        "time_period.start": "2020-09-20T19:03:06+00:00"
    }
}), ('NMX20b', {
    "run": {
        "channels_recorded_auxiliary": [],
        "channels_recorded_electric": [
            "ex",
            "ey"
        ],
        "channels_recorded_magnetic": [
            "hx",
            "hy",
            "hz"
        ],
        "comments": "comments.author:Isaac Sageman; comments.value:X array at 0 deg_\nrotation. All e-lines 50m. Soft sandy dirt. Water tank ~400m NE. County Rd 601 ~200m_\nSE. Warm sunny day.; errors:Found data gaps (2). Gaps of unknown length: 1 [1469160]."
    },
    "data_logger.firmware.author": null,
```

(continues on next page)

(continued from previous page)

```

    "data_logger.firmware.name": null,
    "data_logger.firmware.version": null,
    "data_logger.id": "2612-01",
    "data_logger.manufacturer": "Barry Narod",
    "data_logger.timing_system.drift": 0.0,
    "data_logger.timing_system.type": "GPS",
    "data_logger.timing_system.uncertainty": 0.0,
    "data_logger.type": "NIMS",
    "data_type": "BBMT",
    "id": "NMX20b",
    "sample_rate": 1.0,
    "time_period.end": "2020-10-07T20:28:00+00:00",
    "time_period.start": "2020-09-20T20:12:29+00:00"
}
}])

```

To access a single run you can use either the index of the run or the `run.id`

```
[6]: mt_object.station_metadata.runs[0]
[6]: {
    "run": {
        "channels_recorded_auxiliary": [],
        "channels_recorded_electric": [
            "ex",
            "ey"
        ],
        "channels_recorded_magnetic": [
            "hx",
            "hy",
            "hz"
        ],
        "comments": "comments.author:Isaac Sageman; comments.value:X array at 0 deg\u202c rotation. All e-lines 50m. Soft sandy dirt. Water tank ~400m NE. County Rd 601 ~200m\u202c SE. Warm sunny day.",
        "data_logger.firmware.author": null,
        "data_logger.firmware.name": null,
        "data_logger.firmware.version": null,
        "data_logger.id": "2612-01",
        "data_logger.manufacturer": "Barry Narod",
        "data_logger.timing_system.drift": 0.0,
        "data_logger.timing_system.type": "GPS",
        "data_logger.timing_system.uncertainty": 0.0,
        "data_logger.type": "NIMS",
        "data_type": "BBMT",
        "id": "NMX20a",
        "sample_rate": 1.0,
        "time_period.end": "2020-09-20T19:29:28+00:00",
        "time_period.start": "2020-09-20T19:03:06+00:00"
    }
}
```

```
[7]: mt_object.station_metadata.runs["NMX20b"]

[7]: {
    "run": {
        "channels_recorded_auxiliary": [],
        "channels_recorded_electric": [
            "ex",
            "ey"
        ],
        "channels_recorded_magnetic": [
            "hx",
            "hy",
            "hz"
        ],
        "comments": "comments.author: Isaac Sageman; comments.value:X array at 0 deg_\nrotation. All e-lines 50m. Soft sandy dirt. Water tank ~400m NE. County Rd 601 ~200m_\nSE. Warm sunny day.; errors:Found data gaps (2). Gaps of unknown length: 1 [1469160].]\n",
        "data_logger.firmware.author": null,
        "data_logger.firmware.name": null,
        "data_logger.firmware.version": null,
        "data_logger.id": "2612-01",
        "data_logger.manufacturer": "Barry Narod",
        "data_logger.timing_system.drift": 0.0,
        "data_logger.timing_system.type": "GPS",
        "data_logger.timing_system.uncertainty": 0.0,
        "data_logger.type": "NIMS",
        "data_type": "BBMT",
        "id": "NMX20b",
        "sample_rate": 1.0,
        "time_period.end": "2020-10-07T20:28:00+00:00",
        "time_period.start": "2020-09-20T20:12:29+00:00"
    }
}
```

Channel Metadata

Channel metadata is important because it describes orientation, location, sensors of each channel. These are accessed through the run. Similar to the runs direct access can be through the index or component name.

```
[8]: mt_object.station_metadata.runs[0].channels[0]

[8]: {
    "magnetic": {
        "channel_number": 0,
        "component": "hx",
        "data_quality.rating.value": 0,
        "filter.applied": [
            false
        ],
        "filter.name": [],
        "location.elevation": 0.0,
        "location.latitude": 0.0,
```

(continues on next page)

(continued from previous page)

```

    "location.longitude": 0.0,
    "location.x": 0.0,
    "location.y": 0.0,
    "location.z": 0.0,
    "measurement_azimuth": 9.1,
    "measurement_tilt": 0.0,
    "sample_rate": 0.0,
    "sensor.id": "2509-23",
    "sensor.manufacturer": "Barry Narod",
    "sensor.name": "NIMS",
    "sensor.type": "fluxgate",
    "time_period.end": "2020-09-20T19:29:28+00:00",
    "time_period.start": "2020-09-20T19:03:06+00:00",
    "translated_azimuth": 9.1,
    "type": "magnetic",
    "units": null
  }
}

```

[9]: `mt_object.station_metadata.runs[0].channels["hy"]`

```

[9]: {
  "magnetic": {
    "channel_number": 0,
    "component": "hy",
    "data_quality.rating.value": 0,
    "filter.applied": [
      false
    ],
    "filter.name": [],
    "location.elevation": 0.0,
    "location.latitude": 0.0,
    "location.longitude": 0.0,
    "location.x": 0.0,
    "location.y": 0.0,
    "location.z": 0.0,
    "measurement_azimuth": 99.1,
    "measurement_tilt": 0.0,
    "sample_rate": 0.0,
    "sensor.id": "2509-23",
    "sensor.manufacturer": "Barry Narod",
    "sensor.name": "NIMS",
    "sensor.type": "fluxgate",
    "time_period.end": "2020-09-20T19:29:28+00:00",
    "time_period.start": "2020-09-20T19:03:06+00:00",
    "translated_azimuth": 99.1,
    "type": "magnetic",
    "units": null
  }
}

```

Or you can access the channel metadata through a convenience attribute

```
[10]: mt_object.ex_metadata
[10]: {
    "electric": {
        "channel_number": 0,
        "component": "ex",
        "data_quality.rating.value": 0,
        "dipole_length": 100.0,
        "filter.applied": [
            false
        ],
        "filter.name": [],
        "measurement_azimuth": 9.1,
        "measurement_tilt": 0.0,
        "negative.elevation": 0.0,
        "negative.id": "40201037",
        "negative.latitude": 0.0,
        "negative.longitude": 0.0,
        "negative.manufacturer": "Oregon State University",
        "negative.type": "Pb-PbCl2 kaolin gel Petiau 2 chamber type",
        "negative.x": -50.0,
        "negative.y": 0.0,
        "negative.z": 0.0,
        "positive.elevation": 0.0,
        "positive.id": "40201038",
        "positive.latitude": 0.0,
        "positive.longitude": 0.0,
        "positive.manufacturer": "Oregon State University",
        "positive.type": "Pb-PbCl2 kaolin gel Petiau 2 chamber type",
        "positive.x2": 50.0,
        "positive.y2": 0.0,
        "positive.z2": 0.0,
        "sample_rate": 0.0,
        "time_period.end": "2020-09-20T19:29:28+00:00",
        "time_period.start": "2020-09-20T19:03:06+00:00",
        "translated_azimuth": 9.1,
        "type": "electric",
        "units": null
    }
}
```

Metadata Summary

Metadata is important to keep track of and can be cumbersome, but helps future users of your data to actually use your data. There are a lot of fields here but the most important are ID, location, and timing.

- `mt.survey_metadata.id`
- `mt.station_metadata.id`
- `mt.station_metadata.location.longitude`
- `mt.station_metadata.location.latitude`
- `mt.station_metadata.location.elevation`

- mt.station_metadata.time_period.start
- mt.station_metadata.time_period.end
- mt.station_metadata.transfer_function.processing.parameters
- mt.station_metadata.runs[0].channels[component].measurement_azimuth
- mt.station_metadata.runs[0].channels[component].measurement_tilt
- mt.station_metadata.runs[0].channels[component].translated_azimuth
- mt.station_metadata.runs[0].channels[component].translated_tilt

Data

The most interesting part about a transfer function is the data. This describes how the Earth response to inducing magnetic fields. Under the hood an MT object stores a generic transfer function that has input channels (sources) and output channels (responses) as an `xarray`.

The benefit of using `xarray` is that it has tools for combining various statistical estimates and naturally indexes across a common index. In this case we can index along period for each of the statistical estimates (transfer function, errors, and covariances). The `mt._transfer_function` object is a `xarray.Dataset` and each statistical estimate is an `xarray.DataArray`.

The other benefit is that attributes can be stored directly alongside the arrays for a self describing object. Here we have picked the most important attributes from the metadata to describe the transfer function. These are propagated with each statistical estimate so anytime you retrieve `impedance` or `tipper` the attributes are in the `xarray`.

Note: You'll see below that input channels and output channels have the same components, that's because in order to contain the full transfer function and covariances we need a symmetric matrix. Those components that don't have data are filled with NaNs, which is fine for transfer functions because they are relatively small and not memory intensive.

Generic Transfer Function

```
[11]: mt_object._transfer_function
```

```
[11]: <xarray.Dataset>
Dimensions:                                (output: 5, input: 5, period: 33)
Coordinates:
  * output                               (output) <U2 'ex' 'ey' 'hx' 'hy' 'hz'
  * input                                (input) <U2 'ex' 'ey' 'hx' 'hy' 'hz'
  * period                               (period) float64 4.655 5.818 ... 2.913e+04
Data variables:
  transfer_function                      (period, output, input) complex128 (nan+na...
  transfer_function_error                (period, output, input) float64 nan ... nan
  transfer_function_model_error         (period, output, input) float64 nan ... nan
  inverse_signal_power                 (period, output, input) complex128 (nan+na...
  residual_covariance                  (period, output, input) complex128 (0.0012...
Attributes: (12/14)
  survey:          CONUS South
  project:        USMTArray
  id:             NMX20
  name:            Nations Draw, NM, USA
  latitude:       34.470528
  longitude:      -108.712288
  ...
  datum:           WGS84
```

(continues on next page)

(continued from previous page)

acquired_by:	National Geoelectromagnetic Facility
start:	2020-09-20T19:03:06+00:00
end:	2020-10-07T20:28:00+00:00
runs_processed:	['NMX20a', 'NMX20b']
coordinate_system:	geographic

Errors and Covariances

The generic transfer function xarray also contains the covariances (if provided) and errors of the measured data and also has a place for model errors.

Note: If the tranfer function contains `inverse_signal_power` and `residual_covariance` then the `transfer_function_error` is estimated from these values, otherwise the error from the data file is used.

```
[12]: mt_object._transfer_function.data_vars
```

```
[12]: Data variables:
```

transfer_function	(period, output, input) complex128 (nan+na...)
transfer_function_error	(period, output, input) float64 nan ... nan
transfer_function_model_error	(period, output, input) float64 nan ... nan
inverse_signal_power	(period, output, input) complex128 (nan+na...)
residual_covariance	(period, output, input) complex128 (0.0012...)

MTpy.core.transfer_function.Base

Each of the transfer function estimates are represented by a base class called `mtpy.core.tranfer_function.Base` object which has methods like:

- `inverse` returns the inverse of the transfer function
- `rotate` rotates the transfer function positive clockwise
- `interpolate` interpolates onto a different period index
- `to_xarray` returns an `xarray.DataArray`
- `from_xarray` ingests an `xarray.DataArray`
- `to_dataframe` returns a `pandas.DataFrame` representation of the transfer function indexed by period.
- `from_dataframe` ingests a `pandas.DataFrame`
- `copy` return a copy of the transfer function object
- contains validation methods for inputs

It also has attributes:

- `period` period array
- `frequency` 1/period array
- `comps` components in the transfer function as input channels and output channels
- `n_periods` number of periods

Impedance

The impedance $\hat{\mathbf{Z}}$ describes how the Earth electrically responds to horizontal magnetic fields is formed from this generic transfer function.

$$\tilde{\mathbf{E}}(\omega) = \hat{\mathbf{Z}}(\omega)\tilde{\mathbf{H}}(\omega)$$

To $\hat{\mathbf{Z}}$ is built from the generic transfer function using the horizontal components of the input and output channels. There are two ways to access the impedance from the MT object

- `mt. impedance` will return an xarray of the impedance
- `mt. impedance_error` will return the errors in the impedance measurement
- `mt. impedance_model_error` will return model errors for the impedance
- `mt.Z` will return a `mtpy.core.transfer_function.Z` object which has methods for accessing impedance tensor attributes, which contains the errors.

If you are not sure the transfer function has impedance you can use the method:

```
[13]: mt_object.has_impedance()
```

```
[13]: True
```

MT. impedance

This is an xarray of the impedance and is computed from the generic transfer function using the horizontal components of the input and output channels.

```
[14]: mt_object. impedance
```

```
<xarray.DataArray 'impedance' (period: 33, output: 2, input: 2)>
array([[[[-1.160949e-01-0.2708645j , 3.143284e+00+1.101737j ],
         [-2.470717e+00-0.7784633j , -1.057851e-01+0.1022045j ]],

        [[[-1.051846e-01-0.1912665j , 3.169108e+00+1.007867j ],
         [-2.459892e+00-0.8541335j , -1.325974e-01+0.1473665j ]],

        [[[[-1.289586e-02-0.1937956j , 3.064653e+00+1.063899j ],
          [-2.446347e+00-0.8661013j , -1.222841e-01+0.1580956j ]],

        [[[[-8.208073e-02-0.3117874j , 3.042922e+00+1.006518j ],
          [-2.310078e+00-0.8732821j , -1.620193e-01+0.2110874j ]],

        [[[ 3.353187e-02-0.2855585j , 2.875270e+00+1.083887j ],
          [-2.135730e+00-0.8666129j , -1.881319e-01+0.2420585j ]],

        [[[ 1.014077e-01-0.2989493j , 2.719818e+00+1.103022j ],
          [-2.073182e+00-0.8841784j , -3.048077e-01+0.2344493j ]],

        [[[ 2.441069e-01-0.2828671j , 2.493647e+00+1.179619j ],
          [-1.956353e+00-1.022081j , -3.447069e-01+0.1793671j ]],

        ...[[[ 3.475757e-02+0.0352555j , 1.770043e-01+0.2258684j ],
          [-1.343957e-01-0.1432316j , -4.149757e-02-0.05223549j]]],

        [[[ 2.719606e-02+0.03381913j , 1.438796e-01+0.1859595j ],
```

(continues on next page)

(continued from previous page)

```

[-1.121204e-01-0.1246405j , -3.594606e-02-0.04361913j]],

[[ 2.161943e-02+0.02911898j,  1.137554e-01+0.150974j ],
 [-8.744459e-02-0.103626j , -2.562943e-02-0.03531898j]],

[[ 1.514583e-02+0.02224749j,  8.349043e-02+0.1184608j ],
 [-6.363958e-02-0.08284923j, -2.077583e-02-0.02673749j]],

[[ 9.760046e-03+0.01694262j,  5.923743e-02+0.08950258j],
 [-4.535258e-02-0.06544742j, -1.464005e-02-0.02036262j]],

[[ 1.027061e-02+0.01265869j,  4.057636e-02+0.0674322j ],
 [-3.114365e-02-0.0496578j , -1.030061e-02-0.01919869j]],

[[ 4.834623e-03+0.00983358j,  2.643963e-02+0.05098311j],
 [-2.203037e-02-0.03744689j, -2.953623e-03-0.01293358j]])

```

Coordinates:

```

* output  (output) <U2 'ex' 'ey'
* input   (input) <U2 'hx' 'hy'
* period  (period) float64 4.655 5.818 7.314 ... 1.872e+04 2.913e+04

```

Attributes:

survey:	CONUS South
project:	USMTArray
id:	NMX20
name:	Nations Draw, NM, USA
latitude:	34.470528
longitude:	-108.712288
elevation:	1940.05
declination:	9.09
datum:	WGS84
acquired_by:	National Geoelectromagnetic Facility
start:	2020-09-20T19:03:06+00:00
end:	2020-10-07T20:28:00+00:00
runs_processed:	['NMX20a', 'NMX20b']
coordinate_system:	geographic

MT.Z

The Z object is central to most actions in mtpy and has various attributes that are important to analyzing and visualizing the impedance tensor. Of these are:

- **resistivity** and **phase** are the most common way to represent the impedance tensor. The **resistivity** is an apparent resistivity that describes the volumetric average of the Earth sampled by hemisphere with a radius related to the period through the skin depth. It provides an estimate on how conductive or resistive the subsurface is as a function of period, a proxy for depth. The **phase** is the impedance phase and describes how subsurface resistivity is changing where 45 degrees indicates no change.
 - **res_ij** and **phase_ij** are convenient attributes for accessing only a certain component of the resistivity or phase. If you want to access just the xy component use **res_xy**.
 - **res_det** and **phase_det** represent the resistivity and phase of the determinant of the impedance tensor.
- **phase_tensor** is a different way of representing the impedance tensor that is not effected by near surface distortion like static shift. This returns a **mtpy.core.transfer** function.**Base** object.

```
[15]: mt_object.Z
```

```
[15]: Transfer Function impedance
```

```
-----
Number of periods: 33
Frequency range:      3.43323E-05 -- 2.14844E-01 Hz
Period range:        4.65455E+00 -- 2.91271E+04 s

Has impedance:        True
Has impedance_error:  True
Has impedance_model_error: False
```

```
[16]: mt_object.Z.res_xy
```

```
[16]: array([10.3275702 , 12.8686987 , 15.39508701, 18.78391953, 21.9740757 ,
25.94353835, 29.97081497, 33.72107947, 37.49484018, 39.85170164,
42.22309955, 44.22664358, 46.57708096, 47.53701478, 48.98712171,
50.77007139, 52.33463866, 52.41671744, 51.90545285, 51.26697937,
49.76791648, 46.62343578, 45.61307418, 43.08625057, 40.82330885,
39.53221291, 37.21859669, 34.50456458, 33.45466823, 30.58872947,
27.45312204, 23.19428435, 19.21417312])
```

```
[17]: mt_object.Z.phase_tensor
```

```
[17]: Transfer Function phase_tensor
```

```
-----
Number of periods: 33
Frequency range:      3.43323E-05 -- 2.14844E-01 Hz
Period range:        4.65455E+00 -- 2.91271E+04 s

Has phase_tensor:      True
Has phase_tensor_error: True
Has phase_tensor_model_error: False
```

MT.Tipper

The tipper represents the induction vectors or the Earth's magnetic response to horizontal magnetic fields. You get strong induction vectors when horizontal electrical currents flow through the subsurface like along structural boundaries and faults.

$$H_z(\omega) = \hat{\mathbf{W}}(\omega) \tilde{\mathbf{H}}(\omega)$$

Similar to MT.Z there are methods and attributes for analyzing induction vectors.

- `amplitude` amplitude of the induction vectors
- `phase` phase of the induction vectors
- `angle` horizontal direction of the induction vectors
- `mag` magnitude of the induction vectors

There are also attributes for the real and imaginary components

- `angle_real` and `angle_imag`
- `mag_real` and `mag_imag`

```
[18]: mt_object.Tipper.mag_real
```

```
[18]: array([0.10454066, 0.07782623, 0.08104679, 0.08278517, 0.09892832,
       0.12609043, 0.09750211, 0.07730022, 0.06717719, 0.05336503,
       0.04318178, 0.05157183, 0.07753425, 0.10306461, 0.14128244,
       0.17552209, 0.1995196 , 0.21798128, 0.22319936, 0.22839953,
       0.22834729, 0.22250223, 0.21076025, 0.19728602, 0.17894005,
       0.16590773, 0.14505057, 0.10009294, 0.06102891, 0.06275663,
       0.03127006, 0.14861124, 0.17879201])
```

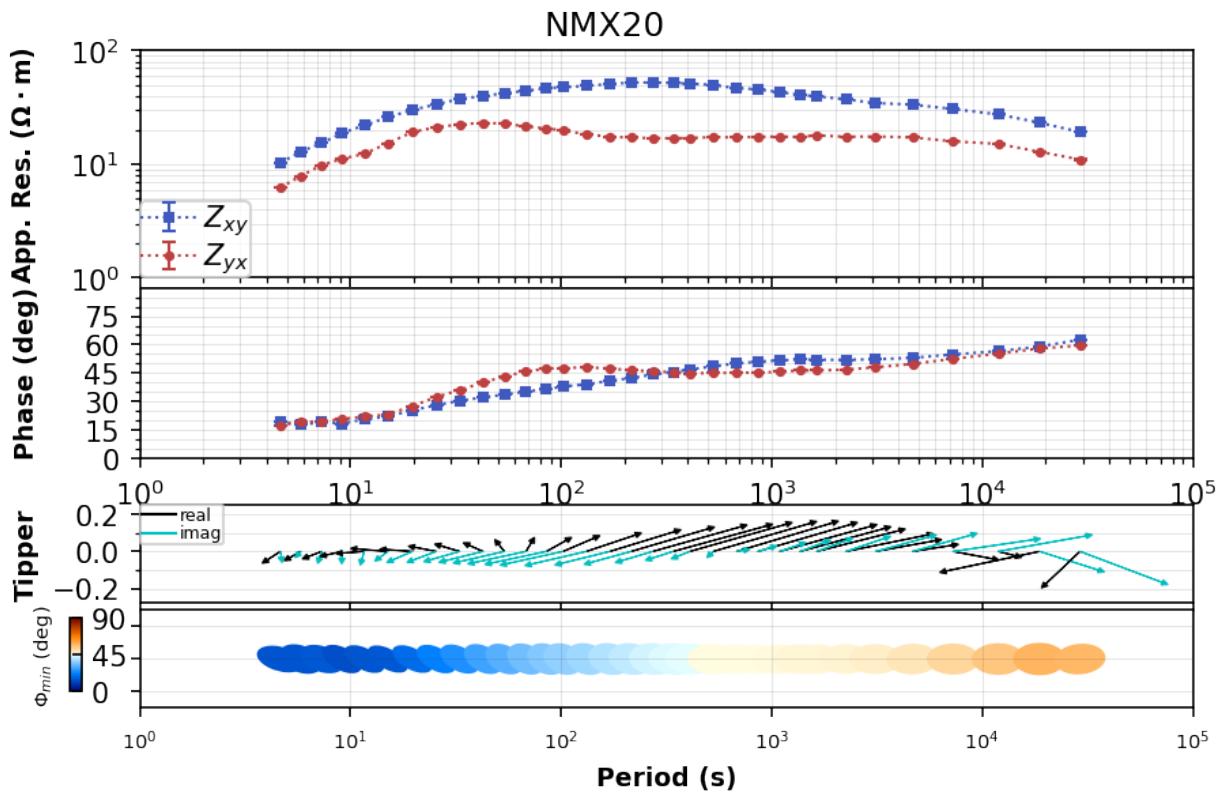
Visualizing

Have a look at the data provides better information than just looking at arrays. There are a few methods for plotting various components of the impedance tensor.

Plot MT Response

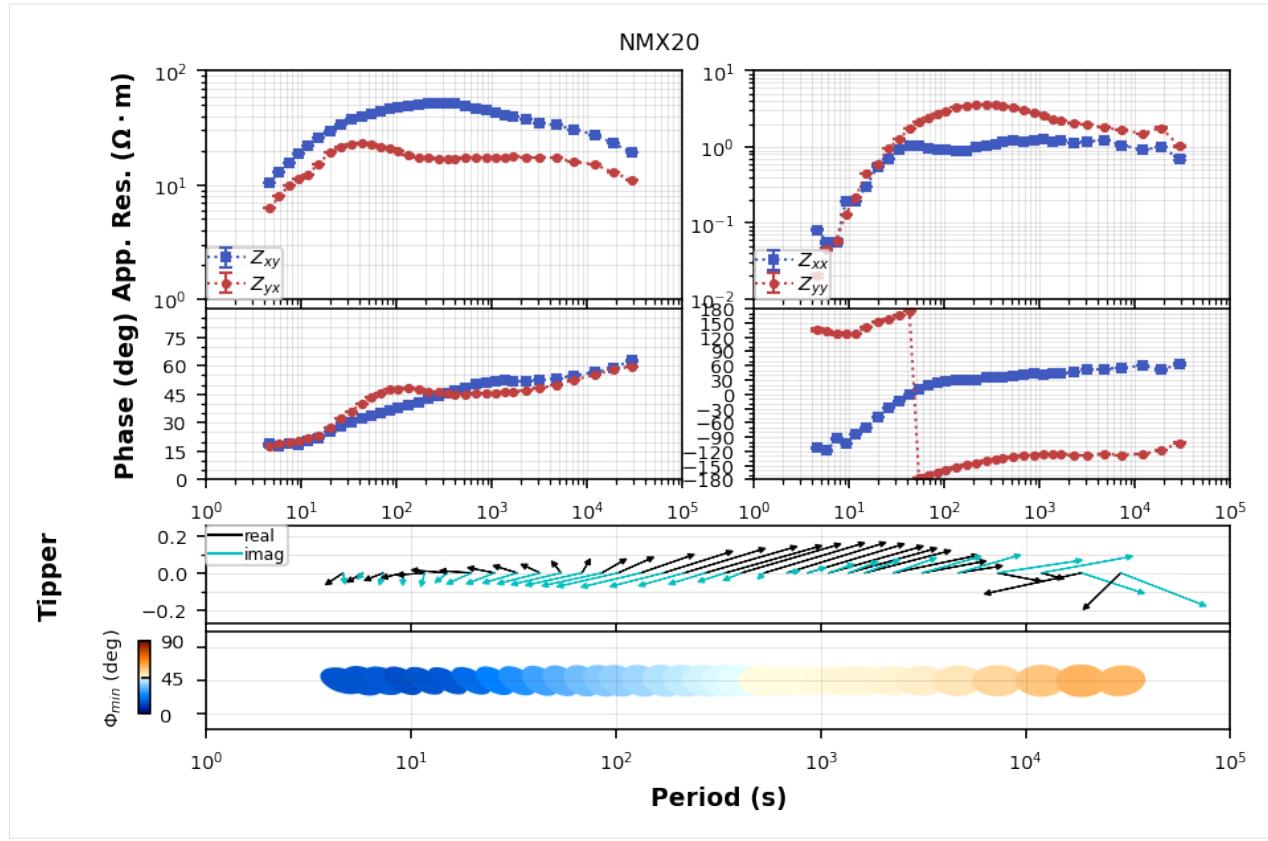
The main visualization is plotting the MT response as apparent resistivity and phase, and if there are induction vectors plot them. This also plots the phase tensor ellipses for completeness. You can turn them on and off if you like.

```
[19]: plot_response = mt_object.plot_mt_response()
```



You can plot all 4 components of the impedance tensor

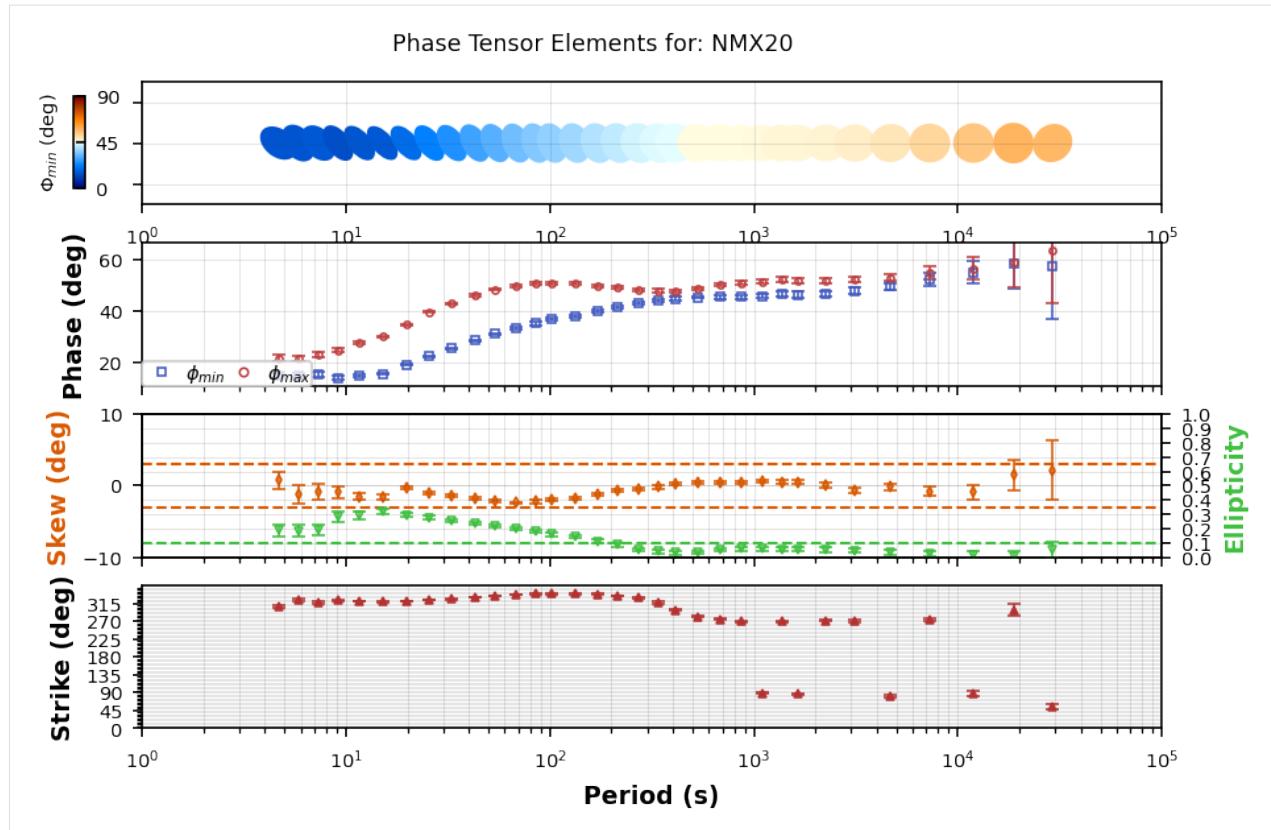
```
[20]: plot_response.plot_num = 2
plot_response.fig_num = 2
plot_response.plot()
```



Plot Phase Tensor component

Sometimes it can be informative to plot the phase tensor components and attributes to determine dimensionality.

```
[21]: plot_pt = mt_object.plot_phase_tensor()
```

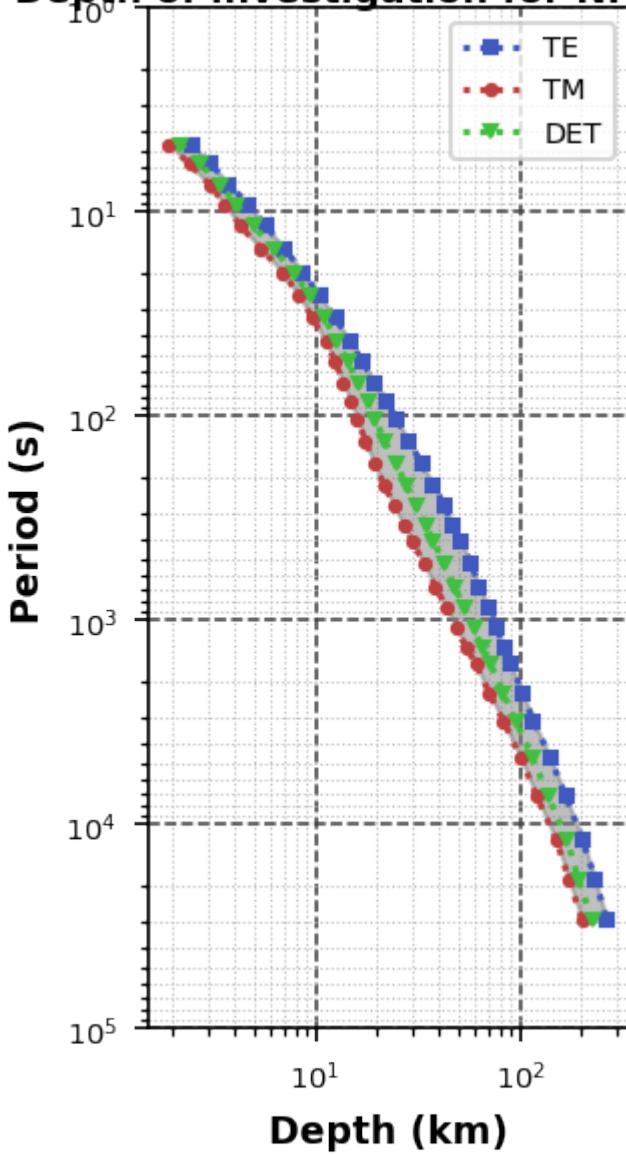


Plot Penetration Depth

Another diagnostic of your data is to estimate the depth of penetration. This is done through a Niblett-Bostick transformation.

```
[22]: plot_nb = mt_object.plot_depth_of_penetration()
```

Depth of investigation for NMX20



Manipulating MT Response

Often you want to manipulate the transfer function by applying a static shift, rotationg, flipping phases, interpolating. There are tools for this.

Rotate

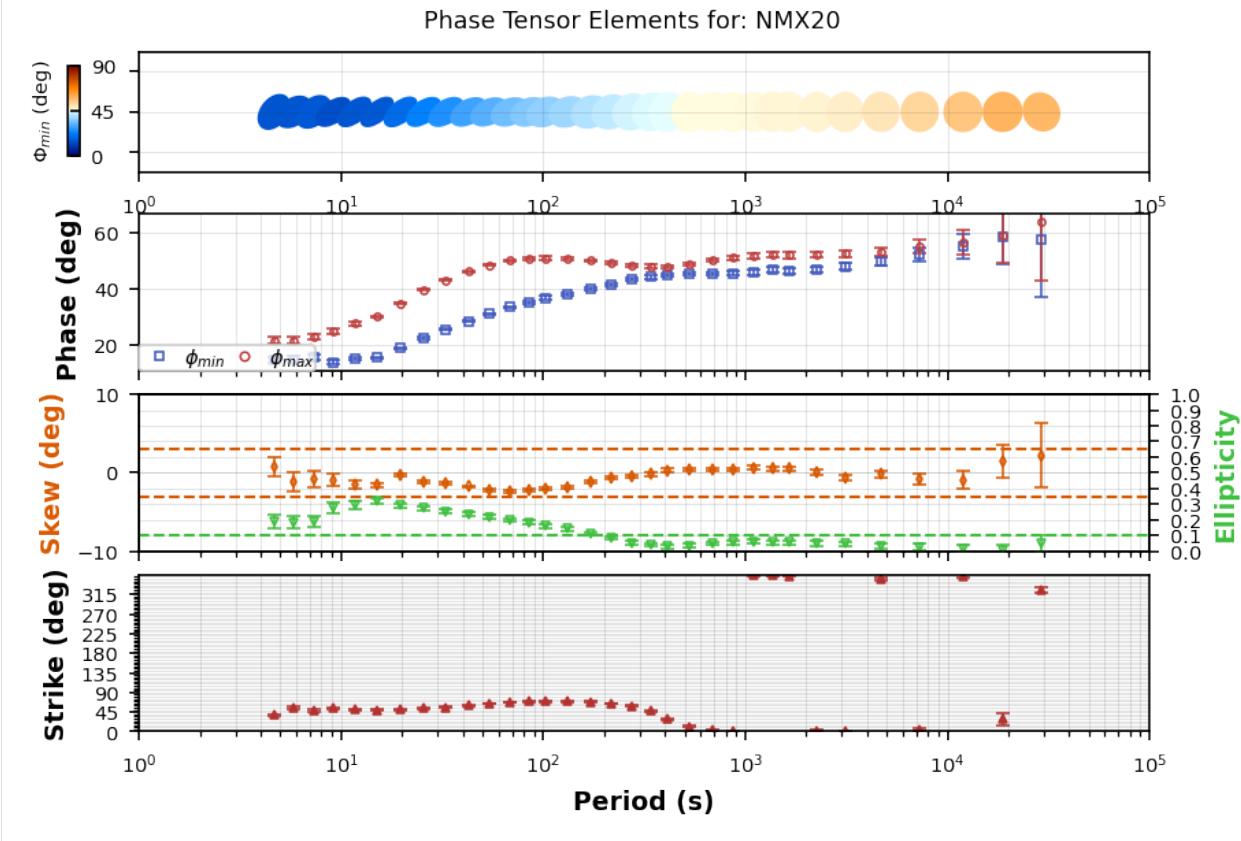
Data rotation helps align the data with geologic structures and optimzing the transfer function to be quasi 2D for modeling.

Geoelectric strike can be estimated from the phase tensor as seen in the plot above. Here this station appears to have a dominant geoelectric strike within the 2D realm is N270E.

Note: All rotations are clockwise assuming that geographic north = 0, and east is 90.

```
[23]: rotated = mt_object.rotate(-270, inplace=False)
rotated.plot_phase_tensor()
```

[23]: Plotting PlotPhaseTensor



Interpolate

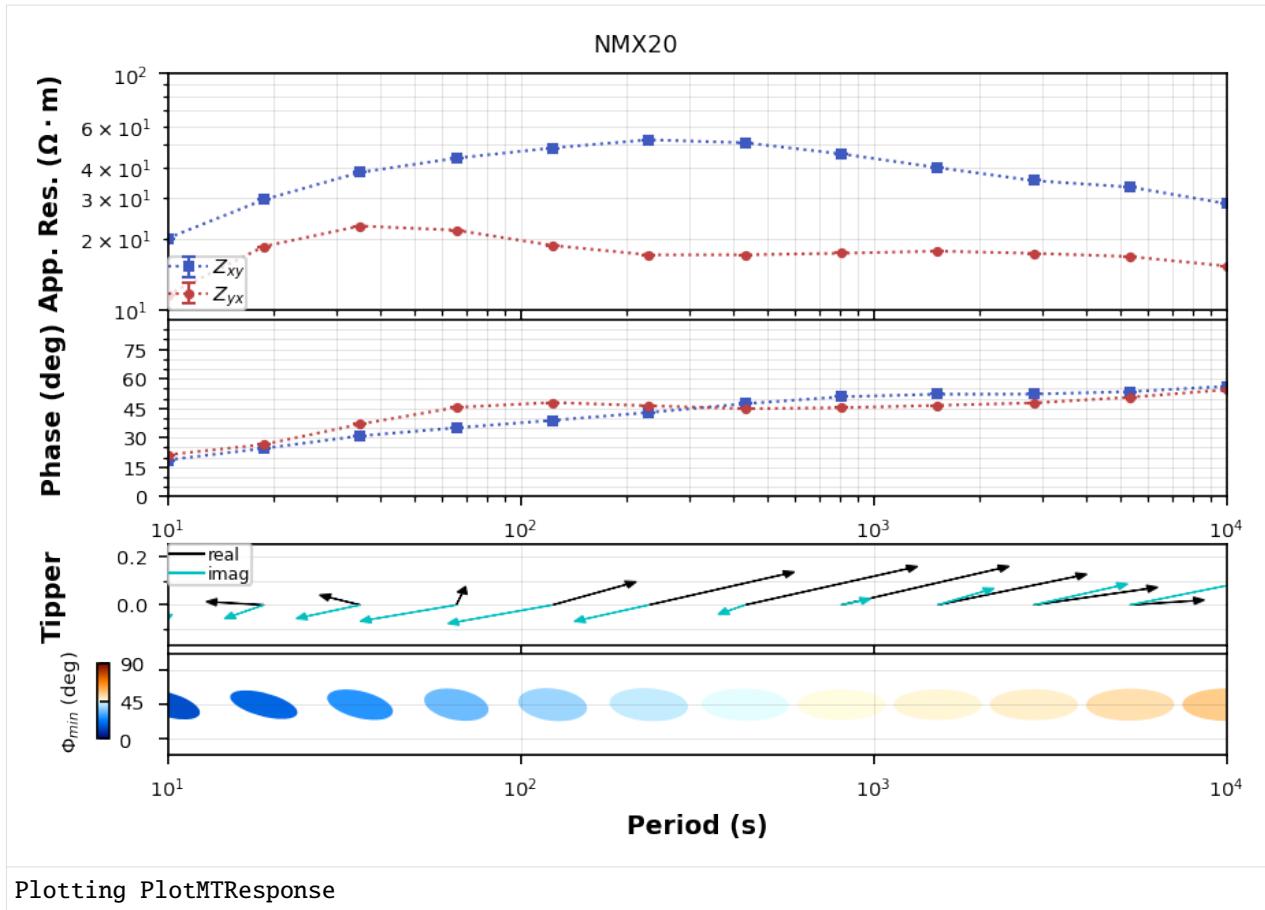
Interpolating data is useful for standardizing a data set for modeling or comparing. Interpolation is done on each component on both real and imaginary parts using a linear interpolation using `scipy.signal.interp1d` internally within `xarray`.

Note: Due to some complexities with `xarray` of removing Nan values a preliminary interpolation is done to interpolate Nan values using the original period range, then an interpolation is done onto the new period range given. You can change the type of interpolation for the ‘na_interpolate’ and ‘method’.

```
[24]: import numpy as np
```

```
[25]: interpolated = mt_object.interpolate(np.logspace(1, 4, 12))
```

```
[26]: interpolated.plot_mt_response()
```



[26]: Plotting PlotMTResponse

Static Shift

Static shift is a distortion that affects the apparent resistivity. We can apply a scalar to shift apparent resistivity up or down.

$$\mathbf{Z} = \mathbf{S} * \mathbf{Z}_0$$

where:

$$\mathbf{S} = \begin{vmatrix} S_x & S_x \\ S_y & S_y \end{vmatrix}$$

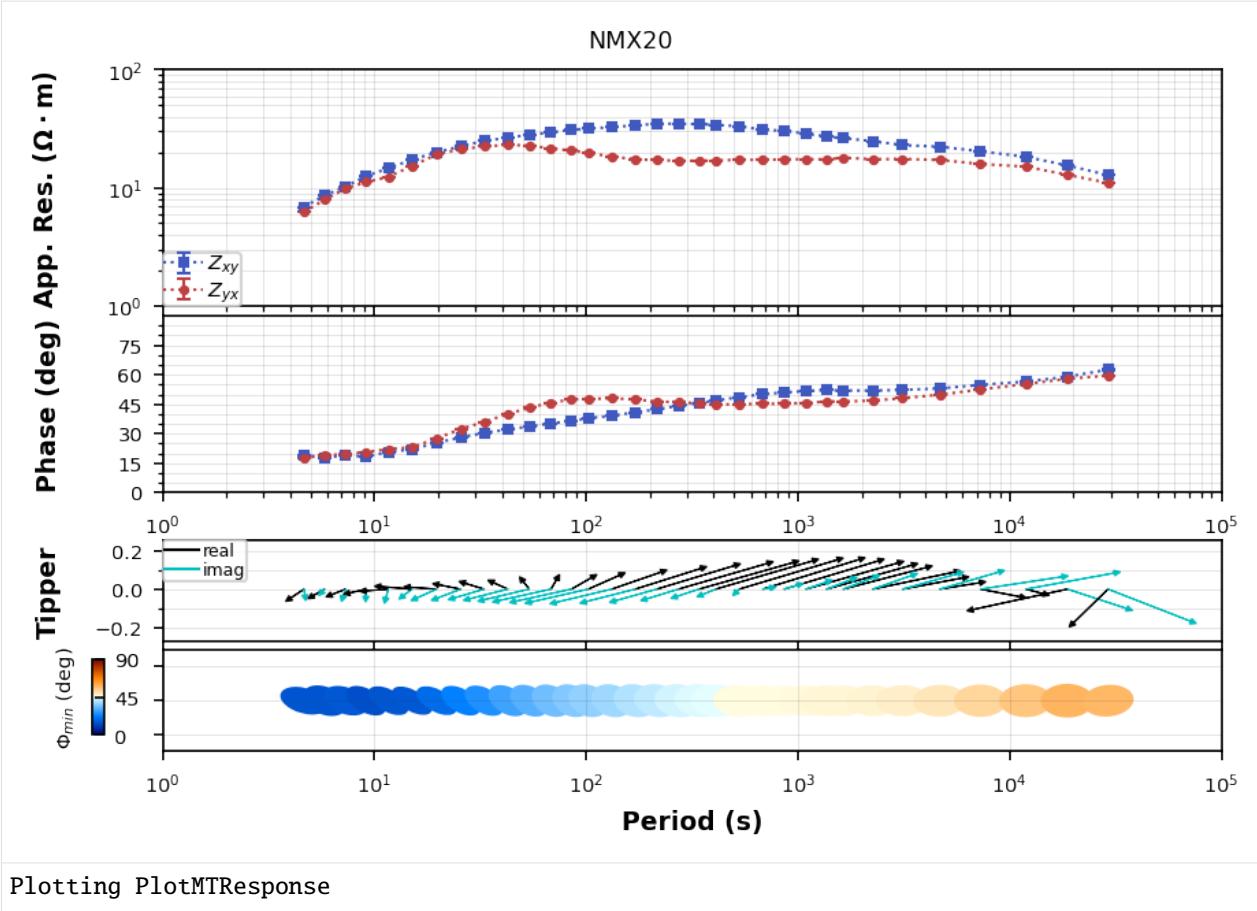
To remove the static shift multiply by \mathbf{S}^{-1}

Note: This assumes that the static shift is given in resistivity coordinates thus the square root of S will be applied.

Note: Numbers great than 1 will move the apparent resistivity down and numbers smaller than 1 will shift apparent resistivity up.

[27]: `static_shifted = mt_object.remove_static_shift(ss_x=1.5)`

[28]: `static_shifted.plot_mt_response()`



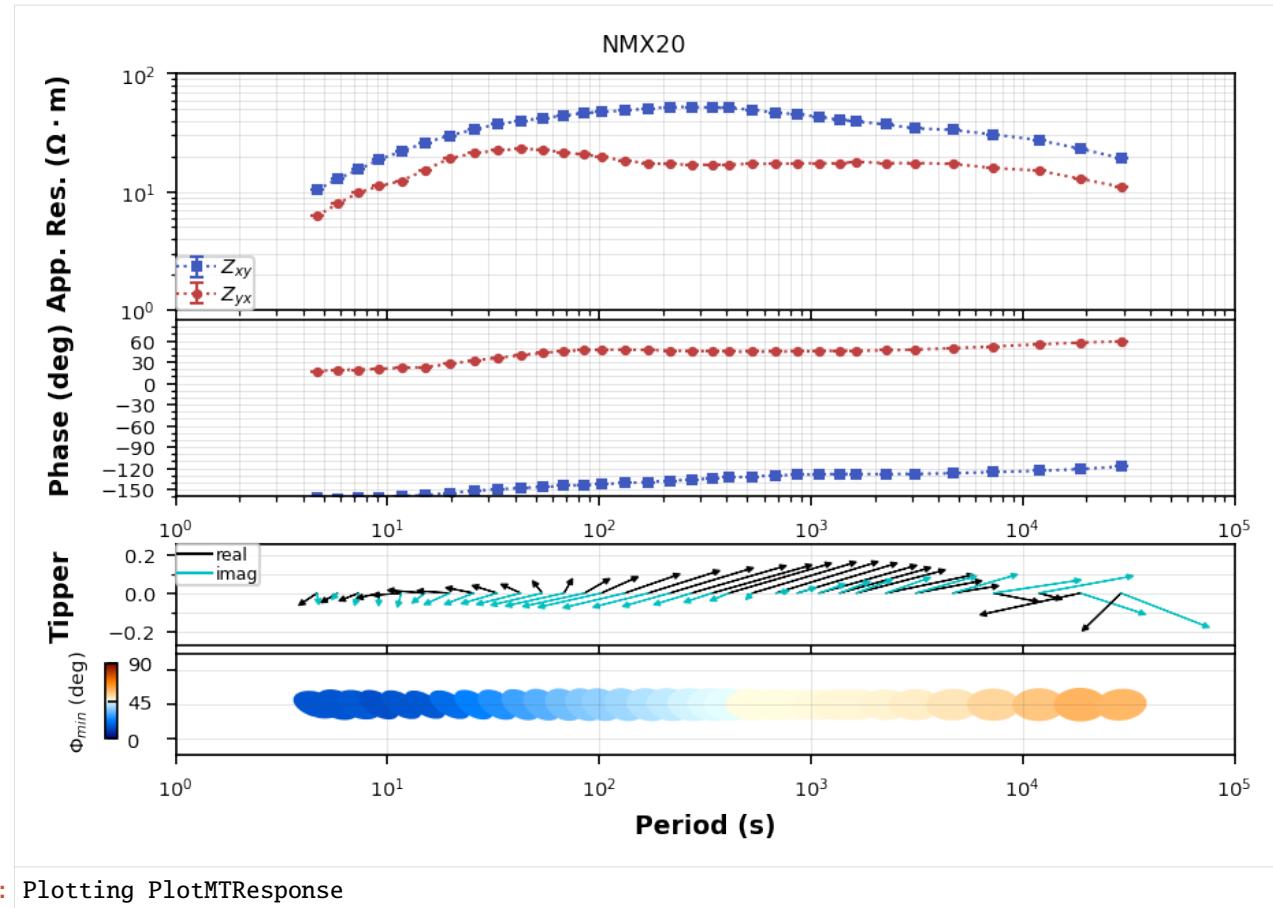
[28]: Plotting PlotMTResponse

Flip Phase

Occasionally you will compute a transfer function that has phase flipped, usually a coil was hooked up backwards, or the electric lines were hooked up in reverse. If Hz data were collected an easy way to tell is plotting induction vectors for the survey and seeing if any look flipped. That is a good indication the magnetic channels was flipped. Otherwise check the electric channels. Here we will assume that the Ex was flipped so we flip Zxx and Zxy.

[29]:

```
flipped = mt_object.flip_phase(zxx=True, zxy=True)
flipped.plot_mt_response()
```

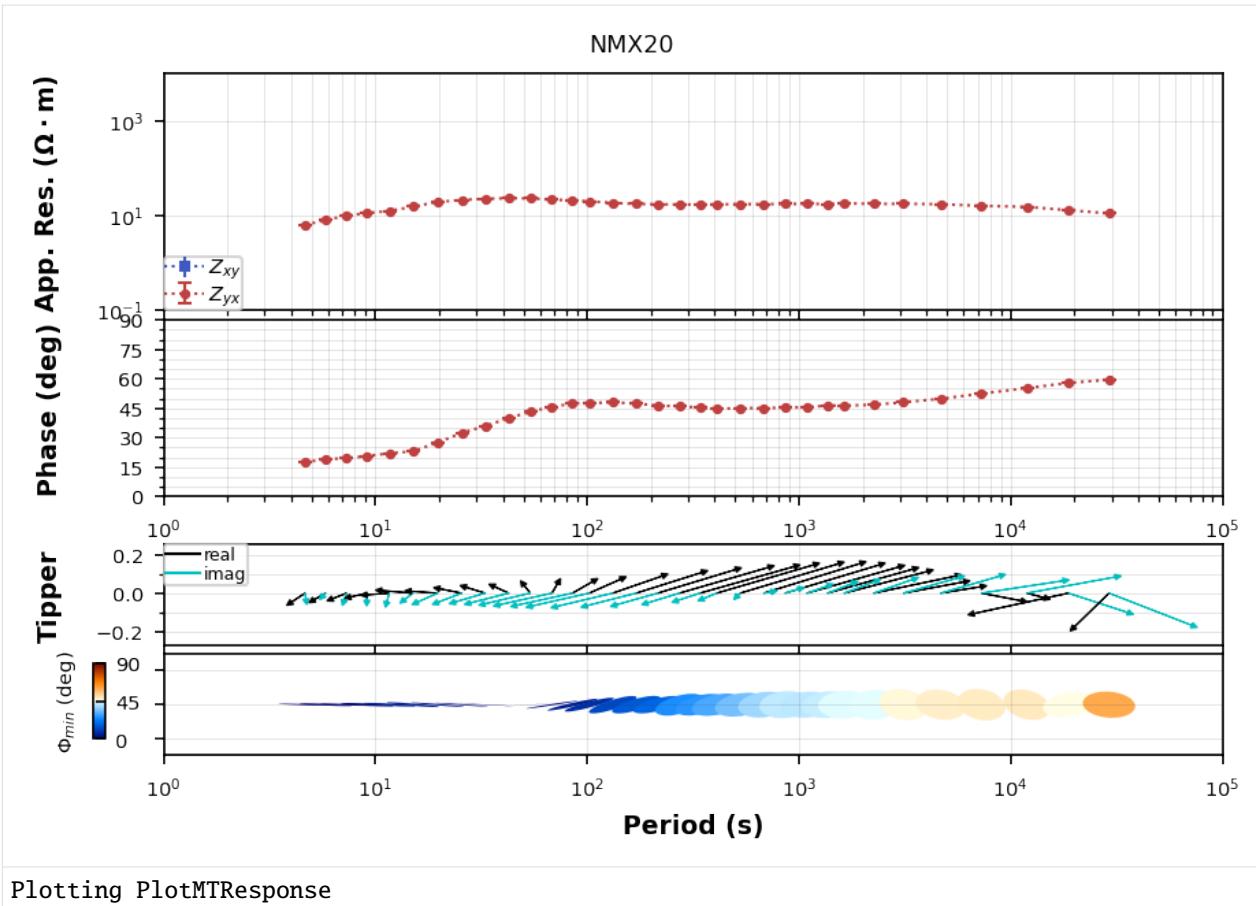


[29]: Plotting PlotMTResponse

Remove a component

Sometimes data can be terrible for a single component and you may want to remove it before analyzing the data.

```
[30]: removed_xy = mt_object.remove_component(zxy=True)
removed_xy.plot_mt_response()
```



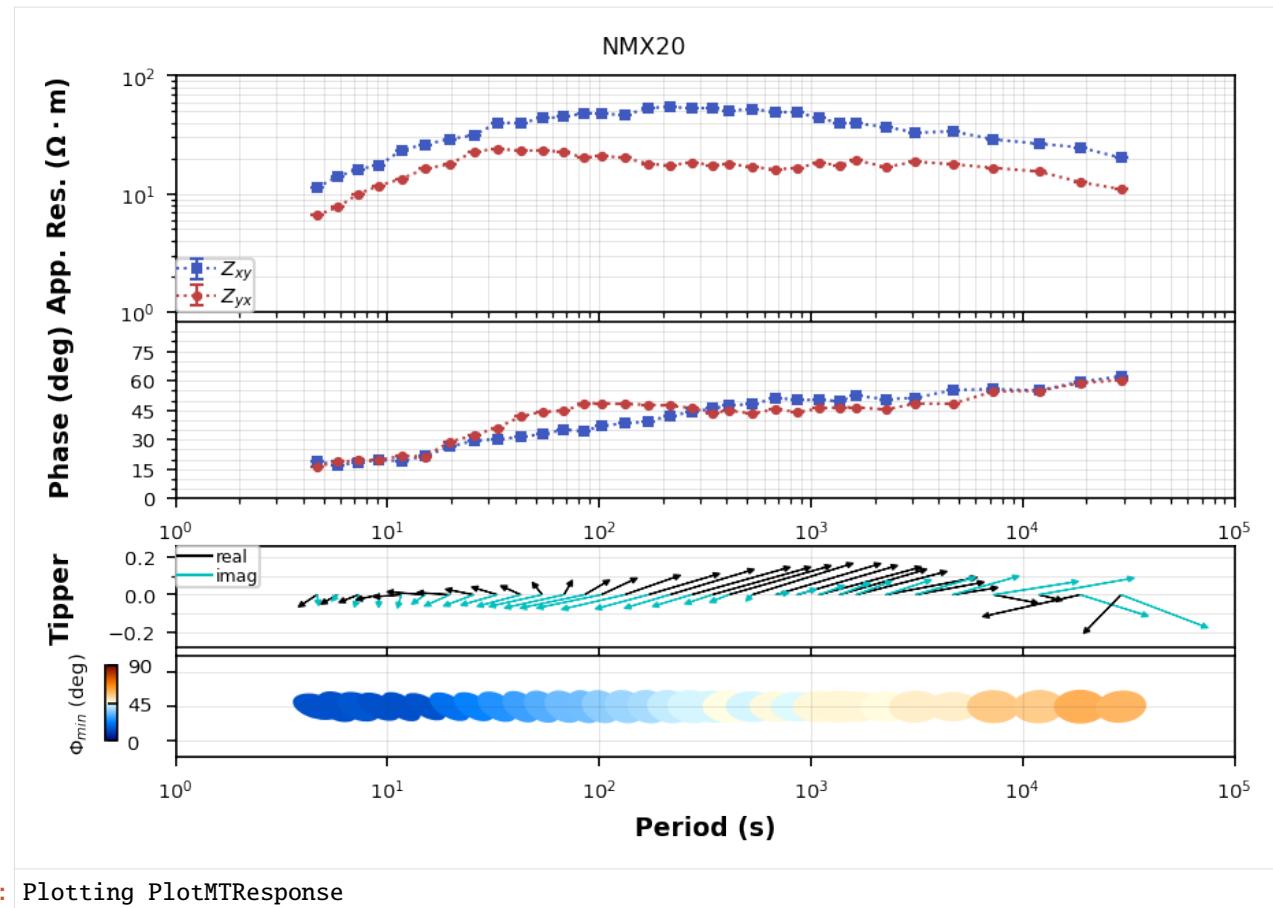
[30]: Plotting PlotMTResponse

Add White Noise

If you are doing some sensitivity tests or inverting synthetic data adding white noise can be useful. Here we will add 5%

[31]: `adding_white_noise = mt_object.add_white_noise(.05, inplace=False)`

[32]: `adding_white_noise.plot_mt_response()`



[32]: Plotting PlotMTResponse

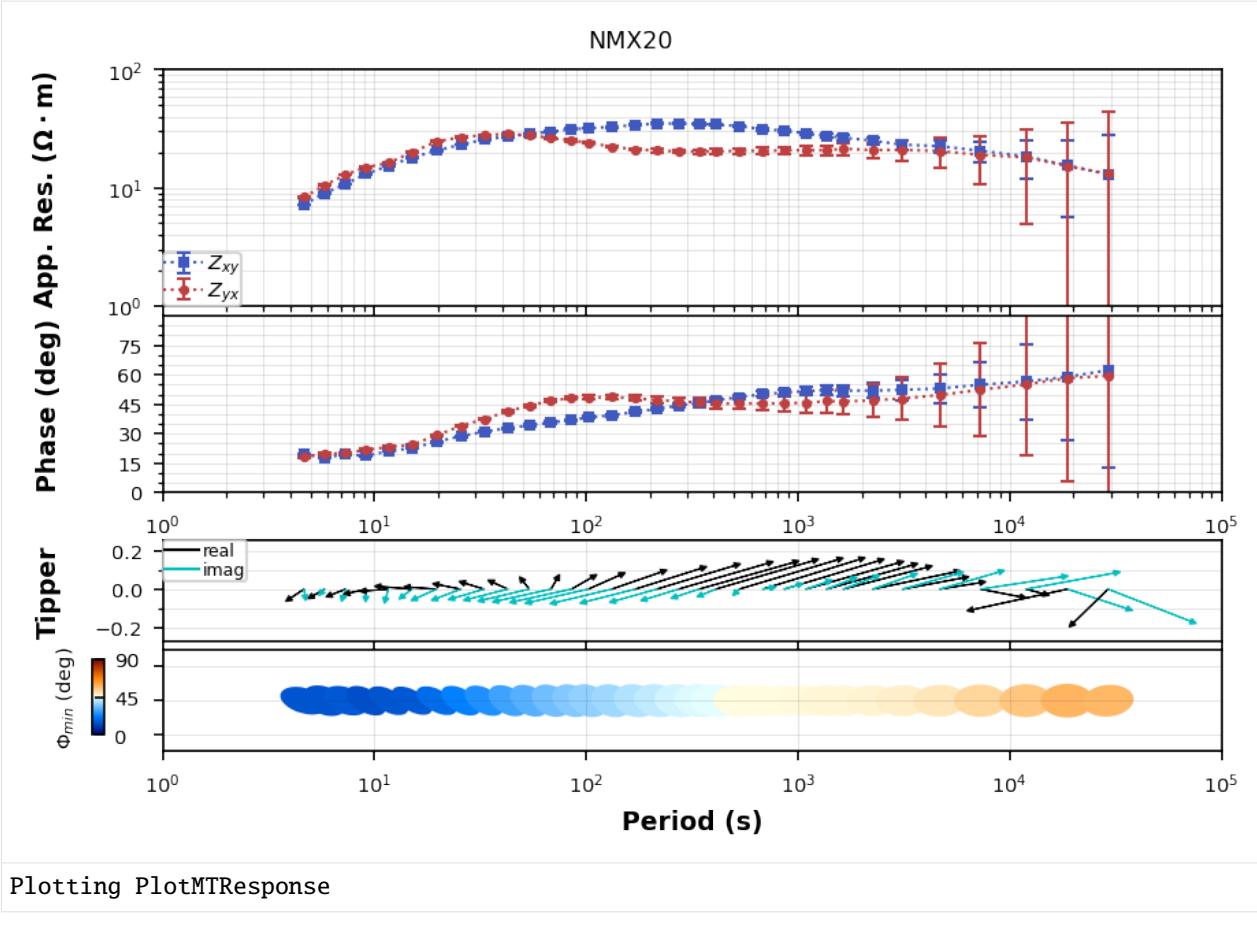
Remove Distortion

Distortion is a pain and usually comes in the form of near surface distortion like a static shift. If you have no other way to estimate distortion, like a TEM measurement or nearby stations a relative estimate can be made using the phase tensor. See Bibby et al., (2005). This basically looks for 1D parts of the transfer function by comparing the TE and TM phases. If the phases match then so should the apparent resistivity. However, knowing the true value of static shift is incomplete and therefore the apparent resistivity curves are pinched together to remove relative static shift.

There might be some issues with uncertainty propagation.

[33]: `distortion_removed = mt_object.remove_distortion(n_frequencies=25)`

[34]: `distortion_removed.plot_mt_response()`



[34]: Plotting PlotMTResponse

[]:

1.2.2 Building an MTCollection

The first step to analyzing a set of transfer functions is to load them into an `MTCollection` which is just a wrapper around an `MTH5` file. The advantage to doing this is that you only have to do this once from the various transfer functions that you have, which might include various flavors of EDI, EMTFXML, J-files, Z-files, AVG-files, etc. The other advantage is that now you have a database to work with: a single file where data is readily accessible vs. multiple ASCII files that need to be read in each time you want to do something.

Using the class `MT` any[1] transfer function can be read into a generic transfer function container `MT`. If you want to adjust survey `.id` or other metadata attributes you can do it from the `MT` object.

1. ^ It works 100% of the time 50% of the time. Most transfer function files are supported but if you find one is not raise an issue

Test Data

Test data can be found at `mtpy-data`. Installation instructions are included in README.

[1]: `from mtpy_data import GRID_LIST, PROFILE_LIST`

1. Initiate MTCollection

First initiate an MTCollection, here we will save to our current working directory.

```
[1]: from pathlib import Path
from mtpy import MT, MTCollection
```

```
[2]: mtc = MTCollection()
```

```
[3]: mtc.open_collection(Path().cwd().joinpath("test_mt_collection.h5"))
```

2. Load Transfer Functions

Step one is locating all the transfer function files you want to read in. Here we will read in a couple different folders. If you have a couple different sets of data from different surveys, the MTCollection will store each in a survey group named by the MT.survey_metadata.id. However, this isn't a common attribute in EDI files, so if you want to separate them out add the correct survey ID.

Note: Loading transfer functions into an MTCollection can take some time so if you have over 100 be patient. But you only have to do this once.

2a. Load Profile Data

The profile data does not have a survey.id in the EDI files so we will add one.

```
[6]: %time
for fn in PROFILE_LIST:
    mt_object = MT()
    mt_object.read(fn)
    mt_object.survey_metadata.id = "profile"
    mtc.add_tf(mt_object)

23:10:19T15:45:54 | INFO | line:122 |mt_metadata.base.metadata | __eq__ | id: 106.001 != 101.001
23:10:19T15:45:54 | INFO | line:122 |mt_metadata.base.metadata | __eq__ | id: 107.001 != 102.001
23:10:19T15:45:54 | WARNING | line:1057 |mth5.mth5 | get_survey | /Experiment/Surveys/profile does not exist, check survey_list for existing names.
23:10:19T15:45:55 | INFO | line:122 |mt_metadata.base.metadata | __eq__ | id: 106.001 != 101.001
23:10:19T15:45:55 | INFO | line:122 |mt_metadata.base.metadata | __eq__ | id: 107.001 != 102.001
23:10:19T15:45:55 | INFO | line:122 |mt_metadata.base.metadata | __eq__ | id: 106.001 != 101.001
23:10:19T15:45:55 | INFO | line:122 |mt_metadata.base.metadata | __eq__ | id: 107.001 != 102.001
23:10:19T15:45:56 | INFO | line:122 |mt_metadata.base.metadata | __eq__ | id: 106.001 != 101.001
23:10:19T15:45:56 | INFO | line:122 |mt_metadata.base.metadata | __eq__ | id: 107.001 != 102.001
23:10:19T15:45:57 | INFO | line:122 |mt_metadata.base.metadata | __eq__ | id: 106.001 != 101.001
23:10:19T15:45:57 | INFO | line:122 |mt_metadata.base.metadata | __eq__ | id: 107.001 != 102.001
```

(continues on next page)

(continued from previous page)

```

23:10:19T15:45:57 | INFO | line:122 |mt_metadata.base.metadata| __eq__ | id: 106.001 !=_
˓→101.001
23:10:19T15:45:57 | INFO | line:122 |mt_metadata.base.metadata| __eq__ | id: 107.001 !=_
˓→102.001
23:10:19T15:45:58 | INFO | line:122 |mt_metadata.base.metadata| __eq__ | id: 106.001 !=_
˓→101.001
23:10:19T15:45:58 | INFO | line:122 |mt_metadata.base.metadata| __eq__ | id: 107.001 !=_
˓→102.001
23:10:19T15:45:59 | INFO | line:122 |mt_metadata.base.metadata| __eq__ | id: 106.001 !=_
˓→101.001
23:10:19T15:45:59 | INFO | line:122 |mt_metadata.base.metadata| __eq__ | id: 107.001 !=_
˓→102.001
23:10:19T15:46:00 | INFO | line:122 |mt_metadata.base.metadata| __eq__ | id: 106.001 !=_
˓→101.001
23:10:19T15:46:00 | INFO | line:122 |mt_metadata.base.metadata| __eq__ | id: 107.001 !=_
˓→102.001
23:10:19T15:46:02 | INFO | line:122 |mt_metadata.base.metadata| __eq__ | id: 106.001 !=_
˓→101.001
23:10:19T15:46:02 | INFO | line:122 |mt_metadata.base.metadata| __eq__ | id: 107.001 !=_
˓→102.001
23:10:19T15:46:03 | INFO | line:122 |mt_metadata.base.metadata| __eq__ | id: 106.001 !=_
˓→101.001
23:10:19T15:46:03 | INFO | line:122 |mt_metadata.base.metadata| __eq__ | id: 107.001 !=_
˓→102.001
23:10:19T15:46:04 | INFO | line:122 |mt_metadata.base.metadata| __eq__ | id: 106.001 !=_
˓→101.001
23:10:19T15:46:04 | INFO | line:122 |mt_metadata.base.metadata| __eq__ | id: 107.001 !=_
˓→102.001
Wall time: 11.7 s

```

2b. Load Grid Data

The grid data also does not have a `survey.id` so we will add one.

```
[7]: %time
for fn in GRID_LIST:
    mt_object = MT()
    mt_object.read(fn)
    mt_object.survey_metadata.id = "grid"
    mtc.add_tf(mt_object)

23:10:19T15:46:29 | WARNING | line:1057 |mth5.mth5 | get_survey | /Experiment/Surveys/
˓→grid does not exist, check survey_list for existing names.
Wall time: 2min 38s
```

3. Working and Master Dataframes

MTCollection includes a summary table of the transfer functions that it contains in the form of a `pandas.DataFrame`, this is the `MTCollection.master_dataframe`. There is also a `MTCollection.working_dataframe` which is a subset of the `master_dataframe` that the user can specify. This allows the MTCollection to be a catch-all for all your transfer functions and then when you only want to work with a small subset from a certain survey or geographic area you can query the `master_dataframe` to set the `working_dataframe`.

3a. Profile Working Dataframe

Here we will choose to work only with data that has a `survey.id = 'profile'`.

```
[4]: mtc.working_dataframe = mtc.master_dataframe.loc[mtc.master_dataframe.survey == "profile"]
```

```
[5]: mtc.working_dataframe
```

	station	survey	latitude	longitude	elevation	tf_id	units	\
59	15125A	profile	-22.370806	149.188639	200.0	15125A	none	
60	15126A	profile	-22.370639	149.193500	200.0	15126A	none	
61	15127A	profile	-22.371028	149.198417	201.0	15127A	none	
62	15128A	profile	-22.370861	149.203306	200.0	15128A	none	
63	15129A	profile	-22.371083	149.208083	202.0	15129A	none	
64	15130A	profile	-22.371222	149.212972	201.0	15130A	none	
65	16122A	profile	-22.325611	149.174361	210.0	16122A	none	
66	16123A	profile	-22.325556	149.179056	213.0	16123A	none	
67	16124A	profile	-22.325694	149.184472	212.0	16124A	none	
68	16125A	profile	-22.325750	149.189306	219.0	16125A	none	
69	16126A	profile	-22.325806	149.194000	214.0	16126A	none	
70	16127A	profile	-22.325889	149.198861	220.0	16127A	none	
	has_impedance	has_tipper	has_covariance	period_min	period_max	\		
59	True	True	False	0.000096	2.857143			
60	True	True	False	0.000096	2.857143			
61	True	True	False	0.000096	2.857143			
62	True	True	False	0.000096	2.857143			
63	True	True	False	0.000096	2.857143			
64	True	True	False	0.000096	2.857143			
65	True	True	False	0.000096	2.857143			
66	True	True	False	0.000096	2.857143			
67	True	True	False	0.000096	2.857143			
68	True	True	False	0.000096	2.857143			
69	True	True	False	0.000096	2.857143			
70	True	True	False	0.000096	2.857143			
	hdf5_reference	station_hdf5_reference						
59	<HDF5 object reference>	<HDF5 object reference>						
60	<HDF5 object reference>	<HDF5 object reference>						
61	<HDF5 object reference>	<HDF5 object reference>						
62	<HDF5 object reference>	<HDF5 object reference>						
63	<HDF5 object reference>	<HDF5 object reference>						
64	<HDF5 object reference>	<HDF5 object reference>						
65	<HDF5 object reference>	<HDF5 object reference>						
66	<HDF5 object reference>	<HDF5 object reference>						
67	<HDF5 object reference>	<HDF5 object reference>						
68	<HDF5 object reference>	<HDF5 object reference>						
69	<HDF5 object reference>	<HDF5 object reference>						
70	<HDF5 object reference>	<HDF5 object reference>						

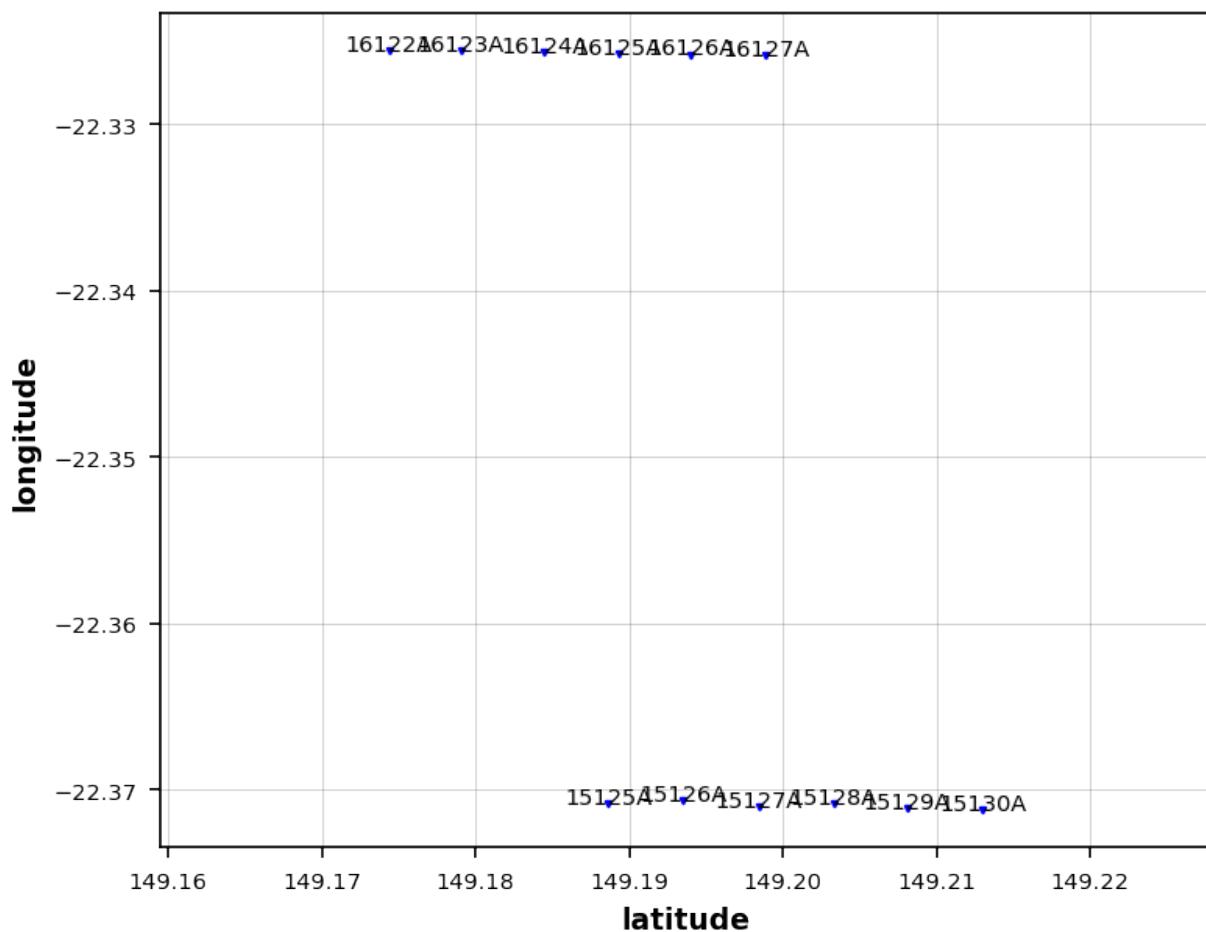
Plot station locations

Now that we have queried for only those stations in the ‘profile’ survey lets plot the station locations just for a sanity check. Looks like the 1600 line and 1500 line are different, so let’s pick just the 1500 line.

Warning: Southern hemisphere locations have issues with contextily and not sure why. Below is an example on how to change the provider. See contextily providers for more details.

```
[8]: stations_plot = mtc.plot_stations(pad=.0001)

23:10:19T16:00:43 | WARNING | line:170 |mtipy.imaging.plot_stations | plot | Could not_
→add base map because Tile URL resulted in a 404 error. Double-check your tile url:_
https://basemap.nationalmap.gov/arcgis/rest/services/USGSTopo/MapServer/tile/15/18469/
→29962
```



Extract only the 15 line

```
[9]: mtc.working_dataframe = mtc.working_dataframe.query('station.str.startswith("15")')
```

```
[10]: mtc.working_dataframe
```

station	survey	latitude	longitude	elevation	tf_id	units	
59	15125A	profile	-22.370806	149.188639	200.0	15125A	none

(continues on next page)

(continued from previous page)

```

60  15126A profile -22.370639  149.193500      200.0  15126A none
61  15127A profile -22.371028  149.198417      201.0  15127A none
62  15128A profile -22.370861  149.203306      200.0  15128A none
63  15129A profile -22.371083  149.208083      202.0  15129A none
64  15130A profile -22.371222  149.212972      201.0  15130A none

    has_impedance has_tipper has_covariance period_min period_max \
59        True      True        False  0.000096  2.857143
60        True      True        False  0.000096  2.857143
61        True      True        False  0.000096  2.857143
62        True      True        False  0.000096  2.857143
63        True      True        False  0.000096  2.857143
64        True      True        False  0.000096  2.857143

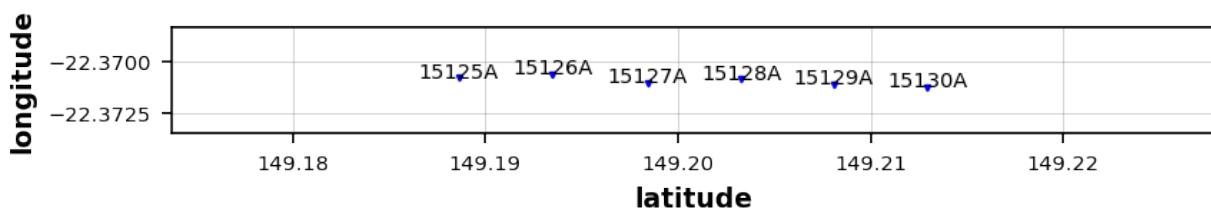
    hdf5_reference station_hdf5_reference
59 <HDF5 object reference> <HDF5 object reference>
60 <HDF5 object reference> <HDF5 object reference>
61 <HDF5 object reference> <HDF5 object reference>
62 <HDF5 object reference> <HDF5 object reference>
63 <HDF5 object reference> <HDF5 object reference>
64 <HDF5 object reference> <HDF5 object reference>

```

```
[11]: station_plot = mtc.plot_stations(pad=.0001)

C:\Users\jpeacock\Anaconda3\envs\em\lib\site-packages\contextily\tile.py:581: UserWarning: The inferred zoom level of 21 is not valid for the current tile provider (valid zooms: 0 - 20).
  warnings.warn(msg)

23:10:19T16:00:54 | WARNING | line:170 |mtpy.imaging.plot_stations | plot | Could not add base map because Tile URL resulted in a 404 error. Double-check your tile url: https://basemap.nationalmap.gov/arcgis/rest/services/USGSTopo/MapServer/tile/20/591168/958827
```



3b. Grid Working DataFrame

Now lets get only the stations in the ‘grid’ survey and plot them.

```
[12]: mtc.working_dataframe = mtc.master_dataframe.loc[mtc.master_dataframe.survey == "grid"]
```

```
[13]: mtc.working_dataframe
```

```
[13]:   station survey  latitude  longitude  elevation  tf_id \
0      gv100     grid  38.611381 -118.535261   1437.400  gv100
```

(continues on next page)

(continued from previous page)

1	gv101	grid	38.594561	-118.351111	1540.550	gv101
2	gv102	grid	38.593692	-118.276822	1554.800	gv102
3	gv103	grid	38.585283	-118.202481	1543.900	gv103
4	gv104	grid	38.596456	-118.136547	1801.800	gv104
5	gv105	grid	38.594131	-118.071933	1901.600	gv105
6	gv106	grid	38.598842	-117.940803	1653.269	gv106
7	gv107	grid	38.612339	-117.882392	1753.700	gv107
8	gv108	grid	38.691575	-118.501819	1904.900	gv108
9	gv109	grid	38.721022	-118.421311	1769.400	gv109
10	gv110	grid	38.672894	-118.330589	1857.100	gv110
11	gv111	grid	38.683950	-118.272311	1893.411	gv111
12	gv112	grid	38.693392	-118.092872	1572.200	gv112
13	gv113	grid	38.660622	-117.991272	1531.200	gv113
14	gv114	grid	38.695694	-117.965139	1542.400	gv114
15	gv115	grid	38.694236	-117.872453	1625.700	gv115
16	gv116	grid	38.772536	-118.499314	1642.800	gv116
17	gv117	grid	38.827428	-118.423014	1505.000	gv117
18	gv118	grid	38.780817	-118.352297	1445.286	gv118
19	gv119	grid	38.793572	-118.284242	1394.190	gv119
20	gv120	grid	38.834919	-118.099611	1304.800	gv120
21	gv121	grid	38.781614	-118.050139	1394.700	gv121
22	gv122	grid	38.779211	-117.957842	1535.500	gv122
23	gv123	grid	38.758194	-117.850214	1855.454	gv123
24	gv124	grid	38.894753	-118.455667	1368.218	gv124
25	gv125	grid	38.897069	-118.379408	1234.500	gv125
26	gv126	grid	38.891994	-118.278522	1234.000	gv126
27	gv127	grid	38.870372	-118.232033	1251.100	gv127
28	gv128	grid	38.887003	-118.080164	1473.400	gv128
29	gv129	grid	38.832450	-118.020161	1357.029	gv129
30	gv130	grid	38.870592	-117.958572	1355.761	gv130
31	gv131	grid	38.899672	-117.858122	1596.800	gv131
32	gv132	grid	38.899553	-118.006981	1381.500	gv132
33	gv133	grid	38.948494	-118.363769	1281.300	gv133
34	gv134	grid	38.964211	-118.309064	1262.900	gv134
35	gv135	grid	38.959308	-118.229806	1236.400	gv135
36	gv136	grid	38.931881	-118.143908	1335.800	gv136
37	gv137	grid	38.938647	-118.040322	1542.100	gv137
38	gv138	grid	38.958664	-117.962525	1431.400	gv138
39	gv139	grid	38.958169	-117.868064	1415.700	gv139
40	gv140	grid	39.065478	-118.467969	1460.400	gv140
41	gv141	grid	39.049328	-118.395528	1620.500	gv141
42	gv142	grid	39.044442	-118.308225	1738.900	gv142
43	gv143	grid	39.053244	-118.252172	1627.200	gv143
44	gv144	grid	39.013067	-118.162347	1548.029	gv144
45	gv145	grid	39.052336	-118.042419	1568.999	gv145
46	gv146	grid	39.045111	-117.961328	1596.200	gv146
47	gv147	grid	39.045894	-117.872028	1587.100	gv147
48	gv148	grid	39.158608	-118.506814	1591.820	gv148
49	gv149	grid	39.128144	-118.421497	1667.600	gv149
50	gv150	grid	39.141736	-118.320472	1490.559	gv150
51	gv151	grid	39.111331	-118.265897	1449.100	gv151
52	gv152	grid	39.141939	-118.155467	1660.881	gv152

(continues on next page)

(continued from previous page)

53	gv153	grid	39.142878	-118.045928	1715.500	gv153	
54	gv154	grid	39.149414	-117.972769	1743.100	gv154	
55	gv155	grid	39.138342	-117.872131	1811.100	gv155	
56	gv160	grid	38.914786	-118.190761	1246.000	gv160	
57	gv161	grid	38.993356	-117.997683	1474.727	gv161	
58	gv163	grid	38.834078	-118.236278	1262.700	gv163	
					units	has_impedance	has_tipper \
0	millivolts_per_kilometer_per_nanotesla					True	True
1	millivolts_per_kilometer_per_nanotesla					True	True
2	millivolts_per_kilometer_per_nanotesla					True	True
3	millivolts_per_kilometer_per_nanotesla					True	True
4	millivolts_per_kilometer_per_nanotesla					True	True
5	millivolts_per_kilometer_per_nanotesla					True	True
6	millivolts_per_kilometer_per_nanotesla					True	True
7	millivolts_per_kilometer_per_nanotesla					True	True
8	millivolts_per_kilometer_per_nanotesla					True	True
9	millivolts_per_kilometer_per_nanotesla					True	True
10	millivolts_per_kilometer_per_nanotesla					True	True
11	millivolts_per_kilometer_per_nanotesla					True	True
12	millivolts_per_kilometer_per_nanotesla					True	True
13	millivolts_per_kilometer_per_nanotesla					True	True
14	millivolts_per_kilometer_per_nanotesla					True	True
15	millivolts_per_kilometer_per_nanotesla					True	True
16	millivolts_per_kilometer_per_nanotesla					True	True
17	millivolts_per_kilometer_per_nanotesla					True	True
18	millivolts_per_kilometer_per_nanotesla					True	True
19	millivolts_per_kilometer_per_nanotesla					True	True
20	millivolts_per_kilometer_per_nanotesla					True	True
21	millivolts_per_kilometer_per_nanotesla					True	True
22	millivolts_per_kilometer_per_nanotesla					True	True
23	millivolts_per_kilometer_per_nanotesla					True	True
24	millivolts_per_kilometer_per_nanotesla					True	True
25	millivolts_per_kilometer_per_nanotesla					True	True
26	millivolts_per_kilometer_per_nanotesla					True	True
27	millivolts_per_kilometer_per_nanotesla					True	True
28	millivolts_per_kilometer_per_nanotesla					True	True
29	millivolts_per_kilometer_per_nanotesla					True	True
30	millivolts_per_kilometer_per_nanotesla					True	True
31	millivolts_per_kilometer_per_nanotesla					True	True
32	millivolts_per_kilometer_per_nanotesla					True	True
33	millivolts_per_kilometer_per_nanotesla					True	True
34	millivolts_per_kilometer_per_nanotesla					True	True
35	millivolts_per_kilometer_per_nanotesla					True	True
36	millivolts_per_kilometer_per_nanotesla					True	True
37	millivolts_per_kilometer_per_nanotesla					True	True
38	millivolts_per_kilometer_per_nanotesla					True	True
39	millivolts_per_kilometer_per_nanotesla					True	True
40	millivolts_per_kilometer_per_nanotesla					True	True
41	millivolts_per_kilometer_per_nanotesla					True	True
42	millivolts_per_kilometer_per_nanotesla					True	True
43	millivolts_per_kilometer_per_nanotesla					True	True

(continues on next page)

(continued from previous page)

44	millivolts_per_kilometer_per_nanotesla		True	True
45	millivolts_per_kilometer_per_nanotesla		True	True
46	millivolts_per_kilometer_per_nanotesla		True	True
47	millivolts_per_kilometer_per_nanotesla		True	True
48	millivolts_per_kilometer_per_nanotesla		True	True
49	millivolts_per_kilometer_per_nanotesla		True	True
50	millivolts_per_kilometer_per_nanotesla		True	True
51	millivolts_per_kilometer_per_nanotesla		True	True
52	millivolts_per_kilometer_per_nanotesla		True	True
53	millivolts_per_kilometer_per_nanotesla		True	True
54	millivolts_per_kilometer_per_nanotesla		True	True
55	millivolts_per_kilometer_per_nanotesla		True	True
56	millivolts_per_kilometer_per_nanotesla		True	True
57	millivolts_per_kilometer_per_nanotesla		True	True
58	millivolts_per_kilometer_per_nanotesla		True	True
0	has_covariance	period_min	period_max	hdf5_reference \
1	False	0.001302	2048.000210	<HDF5 object reference>
2	False	0.001302	2048.000210	<HDF5 object reference>
3	False	0.001302	1365.000061	<HDF5 object reference>
4	False	0.001302	1365.000061	<HDF5 object reference>
5	False	0.001302	1365.000061	<HDF5 object reference>
6	False	0.001302	2048.000210	<HDF5 object reference>
7	False	0.001302	2048.000210	<HDF5 object reference>
8	False	0.001302	2048.000210	<HDF5 object reference>
9	False	0.001302	2048.000210	<HDF5 object reference>
10	False	0.001302	2048.000210	<HDF5 object reference>
11	False	0.001302	2048.000210	<HDF5 object reference>
12	False	0.001302	2048.000210	<HDF5 object reference>
13	False	0.001302	2048.000210	<HDF5 object reference>
14	False	0.001302	2048.000210	<HDF5 object reference>
15	False	0.001302	2048.000210	<HDF5 object reference>
16	False	0.001302	2048.000210	<HDF5 object reference>
17	False	0.001302	2048.000210	<HDF5 object reference>
18	False	0.001302	2048.000210	<HDF5 object reference>
19	False	0.001302	2048.000210	<HDF5 object reference>
20	False	0.001302	2048.000210	<HDF5 object reference>
21	False	0.001302	2048.000210	<HDF5 object reference>
22	False	0.001302	2048.000210	<HDF5 object reference>
23	False	0.001302	2048.000210	<HDF5 object reference>
24	False	0.001302	2048.000210	<HDF5 object reference>
25	False	0.001302	2048.000210	<HDF5 object reference>
26	False	0.001302	2048.000210	<HDF5 object reference>
27	False	0.001302	1115.940903	<HDF5 object reference>
28	False	0.001302	2048.000210	<HDF5 object reference>
29	False	0.001302	2048.000210	<HDF5 object reference>
30	False	0.001302	2048.000210	<HDF5 object reference>
31	False	0.001302	2048.000210	<HDF5 object reference>
32	False	0.001302	2048.000210	<HDF5 object reference>
33	False	0.001302	2048.000210	<HDF5 object reference>
34	False	0.001302	2048.000210	<HDF5 object reference>

(continues on next page)

(continued from previous page)

```

35      False  0.001302 2048.000210 <HDF5 object reference>
36      False  0.001302 2048.000210 <HDF5 object reference>
37      False  0.001302 2048.000210 <HDF5 object reference>
38      False  0.001302 2048.000210 <HDF5 object reference>
39      False  0.001302 2048.000210 <HDF5 object reference>
40      False  0.001302 2048.000210 <HDF5 object reference>
41      False  0.001302 2048.000210 <HDF5 object reference>
42      False  0.001764 2048.000210 <HDF5 object reference>
43      False  0.001302 2048.000210 <HDF5 object reference>
44      False  0.001302 1115.940903 <HDF5 object reference>
45      False  0.001302 2048.000210 <HDF5 object reference>
46      False  0.001302 2048.000210 <HDF5 object reference>
47      False  0.001302 2048.000210 <HDF5 object reference>
48      False  0.001302 2048.000210 <HDF5 object reference>
49      False  0.001302 2048.000210 <HDF5 object reference>
50      False  0.001302 2048.000210 <HDF5 object reference>
51      False  0.001302 2048.000210 <HDF5 object reference>
52      False  0.001302 2048.000210 <HDF5 object reference>
53      False  0.001302 2048.000210 <HDF5 object reference>
54      False  0.001302 2048.000210 <HDF5 object reference>
55      False  0.001302 2048.000210 <HDF5 object reference>
56      False  0.001302 2048.000210 <HDF5 object reference>
57      False  0.001302 1021.063207 <HDF5 object reference>
58      False  0.001302 2048.000210 <HDF5 object reference>

```

```

    station_hdf5_reference
0  <HDF5 object reference>
1  <HDF5 object reference>
2  <HDF5 object reference>
3  <HDF5 object reference>
4  <HDF5 object reference>
5  <HDF5 object reference>
6  <HDF5 object reference>
7  <HDF5 object reference>
8  <HDF5 object reference>
9  <HDF5 object reference>
10 <HDF5 object reference>
11 <HDF5 object reference>
12 <HDF5 object reference>
13 <HDF5 object reference>
14 <HDF5 object reference>
15 <HDF5 object reference>
16 <HDF5 object reference>
17 <HDF5 object reference>
18 <HDF5 object reference>
19 <HDF5 object reference>
20 <HDF5 object reference>
21 <HDF5 object reference>
22 <HDF5 object reference>
23 <HDF5 object reference>
24 <HDF5 object reference>
25 <HDF5 object reference>

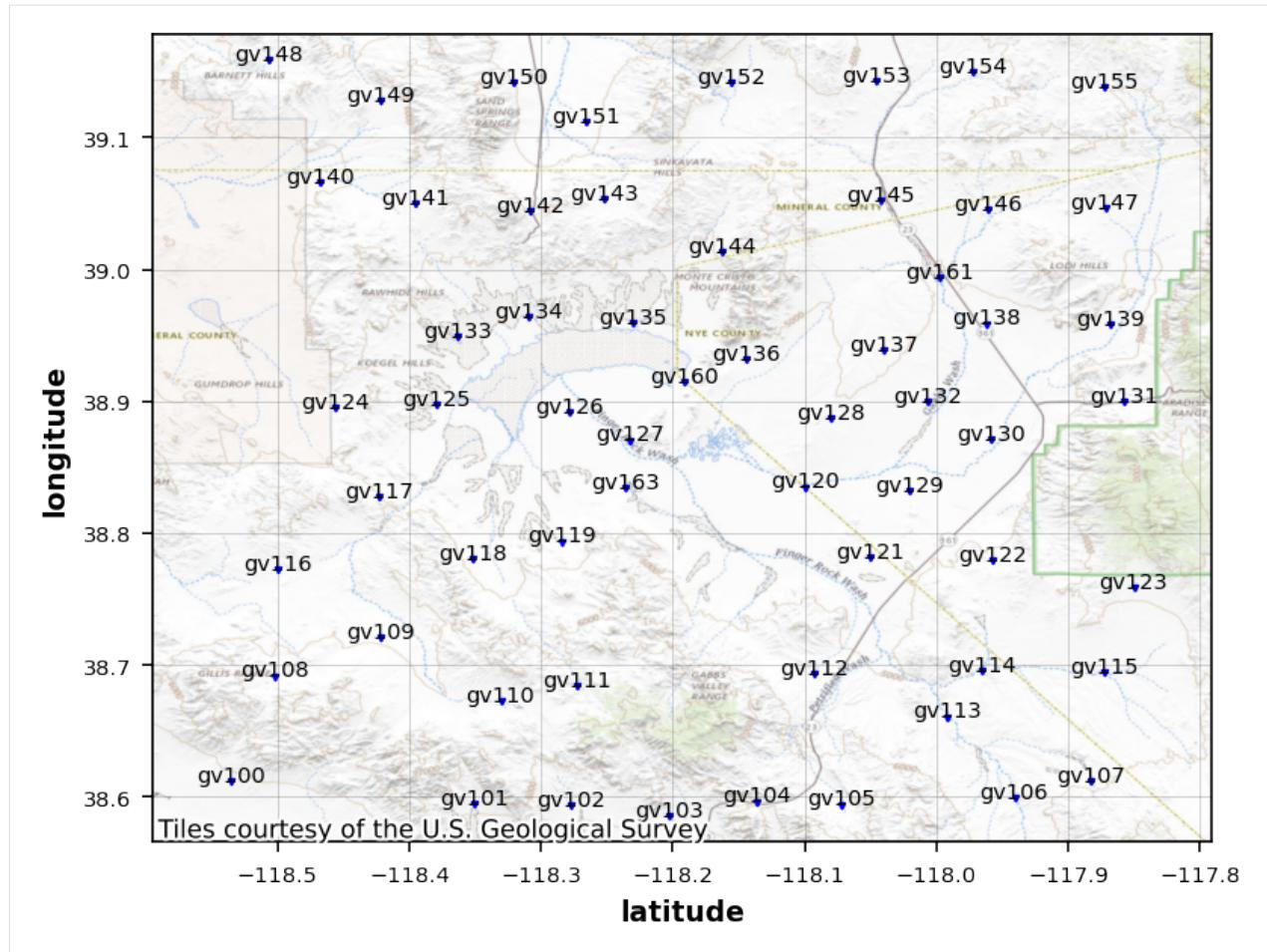
```

(continues on next page)

(continued from previous page)

```
26 <HDF5 object reference>
27 <HDF5 object reference>
28 <HDF5 object reference>
29 <HDF5 object reference>
30 <HDF5 object reference>
31 <HDF5 object reference>
32 <HDF5 object reference>
33 <HDF5 object reference>
34 <HDF5 object reference>
35 <HDF5 object reference>
36 <HDF5 object reference>
37 <HDF5 object reference>
38 <HDF5 object reference>
39 <HDF5 object reference>
40 <HDF5 object reference>
41 <HDF5 object reference>
42 <HDF5 object reference>
43 <HDF5 object reference>
44 <HDF5 object reference>
45 <HDF5 object reference>
46 <HDF5 object reference>
47 <HDF5 object reference>
48 <HDF5 object reference>
49 <HDF5 object reference>
50 <HDF5 object reference>
51 <HDF5 object reference>
52 <HDF5 object reference>
53 <HDF5 object reference>
54 <HDF5 object reference>
55 <HDF5 object reference>
56 <HDF5 object reference>
57 <HDF5 object reference>
58 <HDF5 object reference>
```

```
[17]: station_plot = mtc.plot_stations(pad=.0005)
```



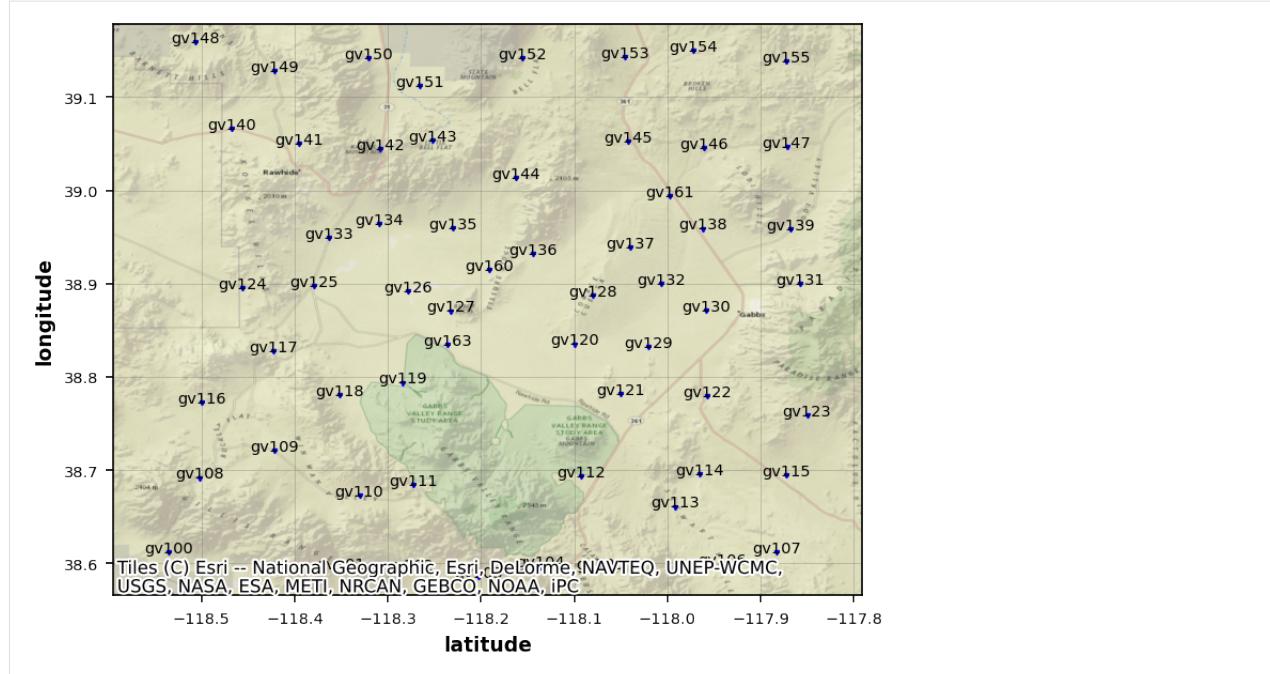
Change Basemap

contextily has good options for basemaps, you can check out all the options at https://contextily.readthedocs.io/en/latest/providers_deepdive.html.

Below is an example of how to change the source basemap to the National Geographic World Map.

```
[18]: import contextily as cx
```

```
[19]: station_plot.cx_source = cx.providers.Esri.NatGeoWorldMap
station_plot.redraw_plot()
```



4. MTCollection vs MTData

MTCollection is the physical object where data are stored in memory and MTData is an extraction of the MTCollection that can be manipulated on RAM. Its designed this way so you don't have to keep accessing the MTH5 file and once loaded into RAM then MTData can manipulate the data in ways that may not want to be permanently stored.

MTCollection does have plotting methods, as seen above, but can be slow for other plotting methods because it needs to load in the data for each plot. This is sort of a bug and needs someone smarter to write better code, but for now convert the `working_dataframe` into an MTData object and from there plots, model inputs, and analysis can be done, which we will see in the MTData example notebook.

MTData can be built similar to MTCollection by reading in transfer function files if you don't want to build an MTH5 file. If you like what you've done with MTData you can write to an MTH5.

```
from mtpy import MT, MTData, MTCollection
md = MTData()

for filename in list_of_tf_files:
    mt_object = MT()
    mt_object.read(filename)
    md.add_station(mt_object)

# Do stuff to MTData object like interpolate, rotate, etc
with MTCollection() as mc:
    mc.open_collection("/path/to/mth5_file.h5")
    mc.from_mt_data(md)
```

5. Close MTCollection

Important: You need to close the MTCollection otherwise the file may get corrupted and you'll have to make the file all over again. Note that once the file is closed the transfer functions are no longer available. Therefore it is wise to convert to an MTData object.

```
[20]: mtc.close_collection()

23:10:19T16:06:23 | INFO | line:760 |mth5.mth5 | close_mth5 | Flushing and closing C:\Users\jpeacock\OneDrive - DOI\Documents\GitHub\mtpy-v2\docs\source\notebooks\test_mt_collection.h5
```

5a. Context Manager

If you just want to build an MTH5 file and then close it the best way to do that is using the context manager:

```
with MTCollection() as mc:
    mc.open_collection("/path/to/mth5_file.h5")
    # add data to the files
```

[]:

1.2.3 MTData: Profile Example

Now that we've created an `MTCollection` object we can use it to do the more interesting things, like analyze strike, plot phase tensors, create inputs for modeling programs.

1. Open Collection

In the previous notebook we created an `MTCollection` object called `test_mt_collection.h5`. Lets open it and get the profile.

```
[1]: from pathlib import Path

from mtpy import MTCollection
```

```
[2]: mtc = MTCollection()
mtc.open_collection(Path().cwd().joinpath("test_mt_collection.h5"))
```

```
[3]: mtc.working_dataframe = mtc.master_dataframe.loc[
    mtc.master_dataframe.survey == "profile"
].query('station.str.startswith("15")')
```

```
[4]: mtc.working_dataframe
```

```
station survey latitude longitude elevation tf_id units \
59 15125A profile -22.370806 149.188639 200.0 15125A none
60 15126A profile -22.370639 149.193500 200.0 15126A none
61 15127A profile -22.371028 149.198417 201.0 15127A none
62 15128A profile -22.370861 149.203306 200.0 15128A none
63 15129A profile -22.371083 149.208083 202.0 15129A none
64 15130A profile -22.371222 149.212972 201.0 15130A none

has_impedance has_tipper has_covariance period_min period_max \
59 True True False 0.000096 2.857143
60 True True False 0.000096 2.857143
61 True True False 0.000096 2.857143
62 True True False 0.000096 2.857143
```

(continues on next page)

(continued from previous page)

63	True	True	False	0.000096	2.857143
64	True	True	False	0.000096	2.857143
		hdf5_reference	station_hdf5_reference		
59	<HDF5 object reference>	<HDF5 object reference>			
60	<HDF5 object reference>	<HDF5 object reference>			
61	<HDF5 object reference>	<HDF5 object reference>			
62	<HDF5 object reference>	<HDF5 object reference>			
63	<HDF5 object reference>	<HDF5 object reference>			
64	<HDF5 object reference>	<HDF5 object reference>			

2. Convert to MTData

Now that we have the profile let's convert it to an MTData object.

```
[5]: mtd = mtc.to_mt_data()
```

2a. Close MTCollection

You can now close the MTCollection to make sure if something crashes the file won't get corrupt for unknown reasons.

```
[6]: mtc.close_collection()
```

```
23:10:23T09:21:00 | INFO | line:760 |mth5.mth5 | close_mth5 | Flushing and closing C:\Users\jpeacock\OneDrive - DOT\Documents\GitHub\mtypy-v2\docs\source\notebooks\test_mt_collection.h5
```

2b. Station Locations

A convenient attribute of MTData is the `station_locations` object. This is a `mtypy.core.MTStations` object and represented as a `pandas.DataFrame`. You will notice here that `east` and `north` are not populated, that is because the MTData object is currently agnostic to a UTM coordinate system.

```
[7]: mtd.station_locations
```

```
[7]:   survey station  latitude  longitude  elevation datum_epsg  east  north  \
0  profile  15125A -22.370806  149.188639    200.0      4326  0.0   0.0
1  profile  15126A -22.370639  149.193500    200.0      4326  0.0   0.0
2  profile  15127A -22.371028  149.198417    201.0      4326  0.0   0.0
3  profile  15128A -22.370861  149.203306    200.0      4326  0.0   0.0
4  profile  15129A -22.371083  149.208083    202.0      4326  0.0   0.0
5  profile  15130A -22.371222  149.212972    201.0      4326  0.0   0.0

   utm_epsg  model_east  model_north  model_elevation  profile_offset
0     None      0.0       0.0        200.0            0.0
1     None      0.0       0.0        200.0            0.0
2     None      0.0       0.0        201.0            0.0
3     None      0.0       0.0        200.0            0.0
4     None      0.0       0.0        202.0            0.0
5     None      0.0       0.0        201.0            0.0
```

2c. Setting UTM CRS

It's important to set the `MTData.utm_crs` attribute to make sure that stations can be projected into meters for plotting and creating model files. You can do this a couple of ways either through the `utm_crs` method or if you know the EPSG number you can input that. They should both do the same if you input the number.

If you have created a custom CRS, be sure to set `mtd.utm_crs` with the custom CRS.

```
[8]: mtd.utm_crs = 32755
```

```
[9]: mtd.utm_crs
```

```
[9]: <Derived Projected CRS: EPSG:32755>
Name: WGS 84 / UTM zone 55S
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: Between 144°E and 150°E, southern hemisphere between 80°S and equator, onshore ↵
and offshore. Australia. Papua New Guinea.
- bounds: (144.0, -80.0, 150.0, 0.0)
Coordinate Operation:
- name: UTM zone 55S
- method: Transverse Mercator
Datum: World Geodetic System 1984 ensemble
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

```
[10]: mtd.station_locations
```

```
[10]:   survey station  latitude  longitude  elevation datum_epsg \
0  profile  15125A -22.370806  149.188639    200.0      4326
1  profile  15126A -22.370639  149.193500    200.0      4326
2  profile  15127A -22.371028  149.198417    201.0      4326
3  profile  15128A -22.370861  149.203306    200.0      4326
4  profile  15129A -22.371083  149.208083    202.0      4326
5  profile  15130A -22.371222  149.212972    201.0      4326

          east        north  utm_epsg  model_east  model_north \
0  725360.330833  7.524490e+06    32755       0.0       0.0
1  725861.314778  7.524502e+06    32755       0.0       0.0
2  726367.124612  7.524451e+06    32755       0.0       0.0
3  726870.972550  7.524462e+06    32755       0.0       0.0
4  727362.745811  7.524430e+06    32755       0.0       0.0
5  727866.099363  7.524408e+06    32755       0.0       0.0

  model_elevation  profile_offset
0            200.0           0.0
1            200.0           0.0
2            201.0           0.0
3            200.0           0.0
4            202.0           0.0
5            201.0           0.0
```

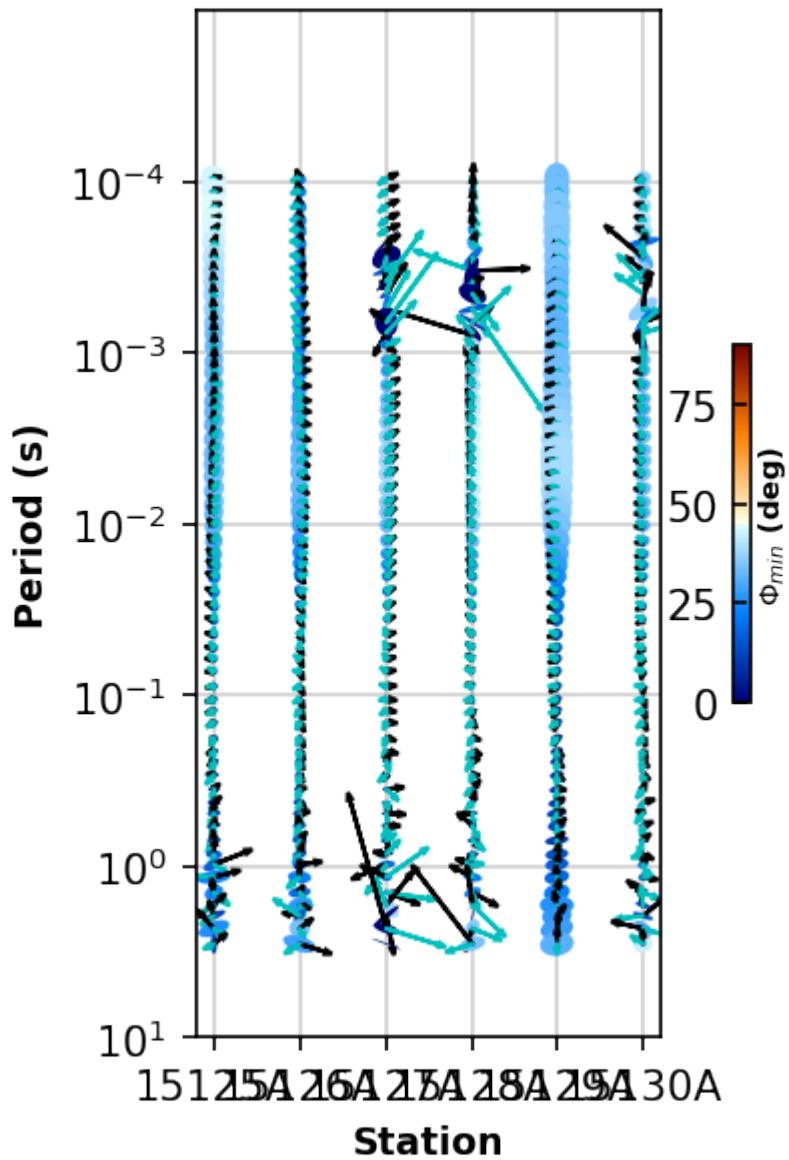
Now you can see that the locations have been projected into the given coordinate system.

Note: The `model_east` and `model_north` do not get populated, those are for relative coordinates for modeling.

3. Plot Phase Tensor Pseudosection

Here we are adjusting the stretch in the x-direction and plotting the tipper vectors ('r' = real, 'i' = imaginary, 'y' = yes to plot)

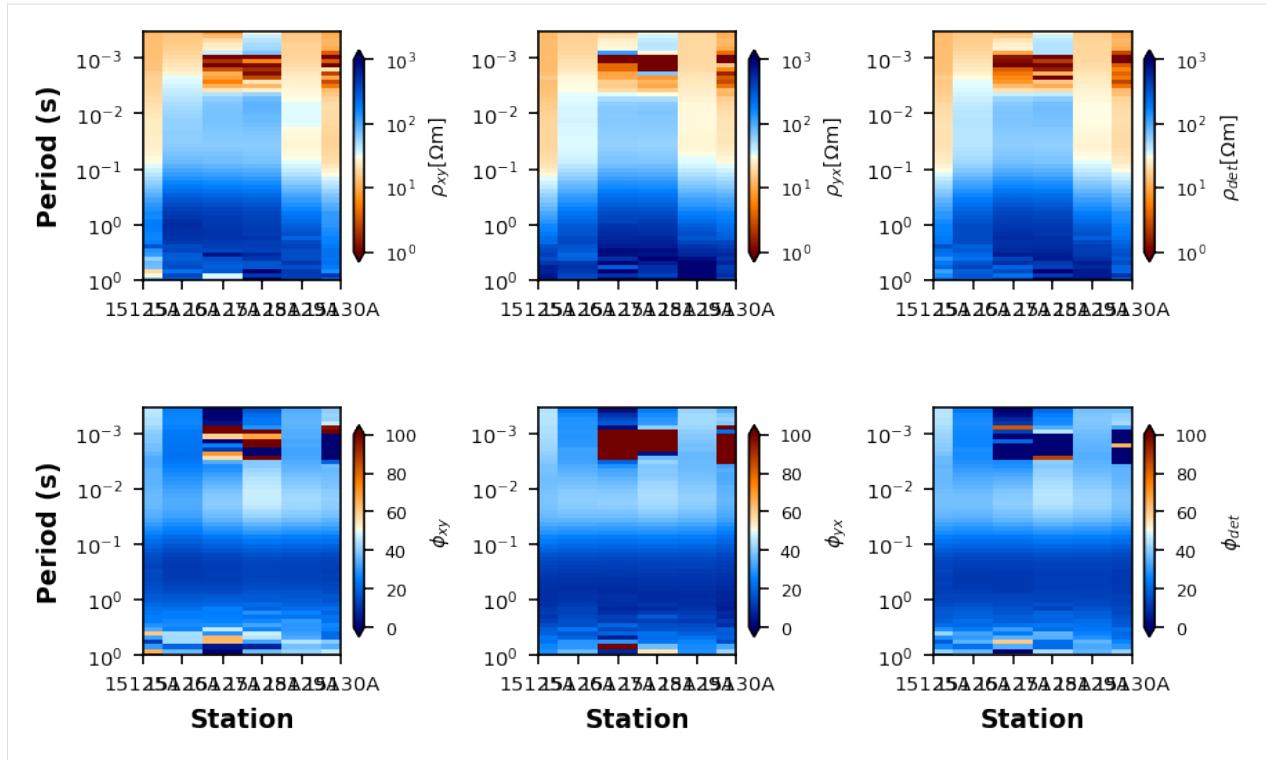
```
[11]: ptps_plot = mtd.plot_phase_tensor_pseudosection(x_stretch=10, plot_tipper="yri")
```



4. Plot Resistivity and Phase Pseudosections

Here we are plotting the xy, yx, and det components of the impedance tensor.

```
[12]: rpps_plot = mtd.plot_resistivity_phase_pseudosections(y_stretch=700, plot_det=True)
```



5. Plot Strike

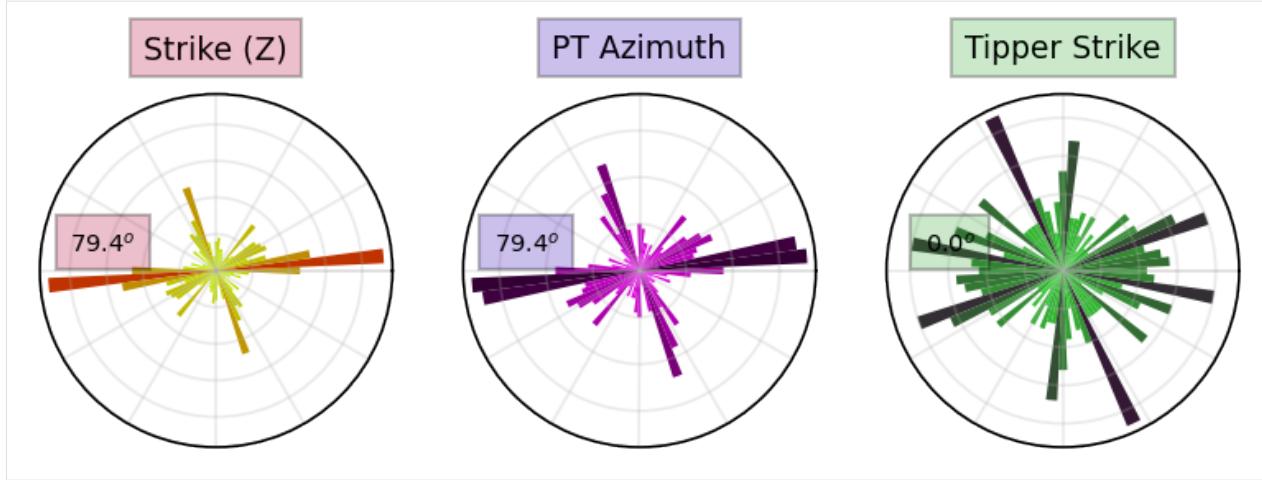
Plotting strike angles are very important when working with profile data. We can plot strike in a couple of ways as a compilation of all strike angles, or per decade. If you really want you could do it by region you query for the stations in each region.

5a. All Periods

Here we will plot all periods of estimated strike. Notice that the plot includes the strike as estimated from the invariants (left) of [Weaver et al. \(2002\)](#), the phase tensor (middle) of [Caldwell et al. \(2004\)](#), and the induction vector strike.

Important: The induction strike points towards good conductors so should therefore be perpendicular to the impedance strike. We left it this way as a sanity check on strike angles.

```
[13]: strike_plot_all = mtd.plot_strike()
23:10:23T09:21:29 | INFO | line:892 |mtipy.imaging.plot_strike | _plot_all_periods | Note:
→ North is assumed to be 0 and the strike angle is measured clockwise positive.
```

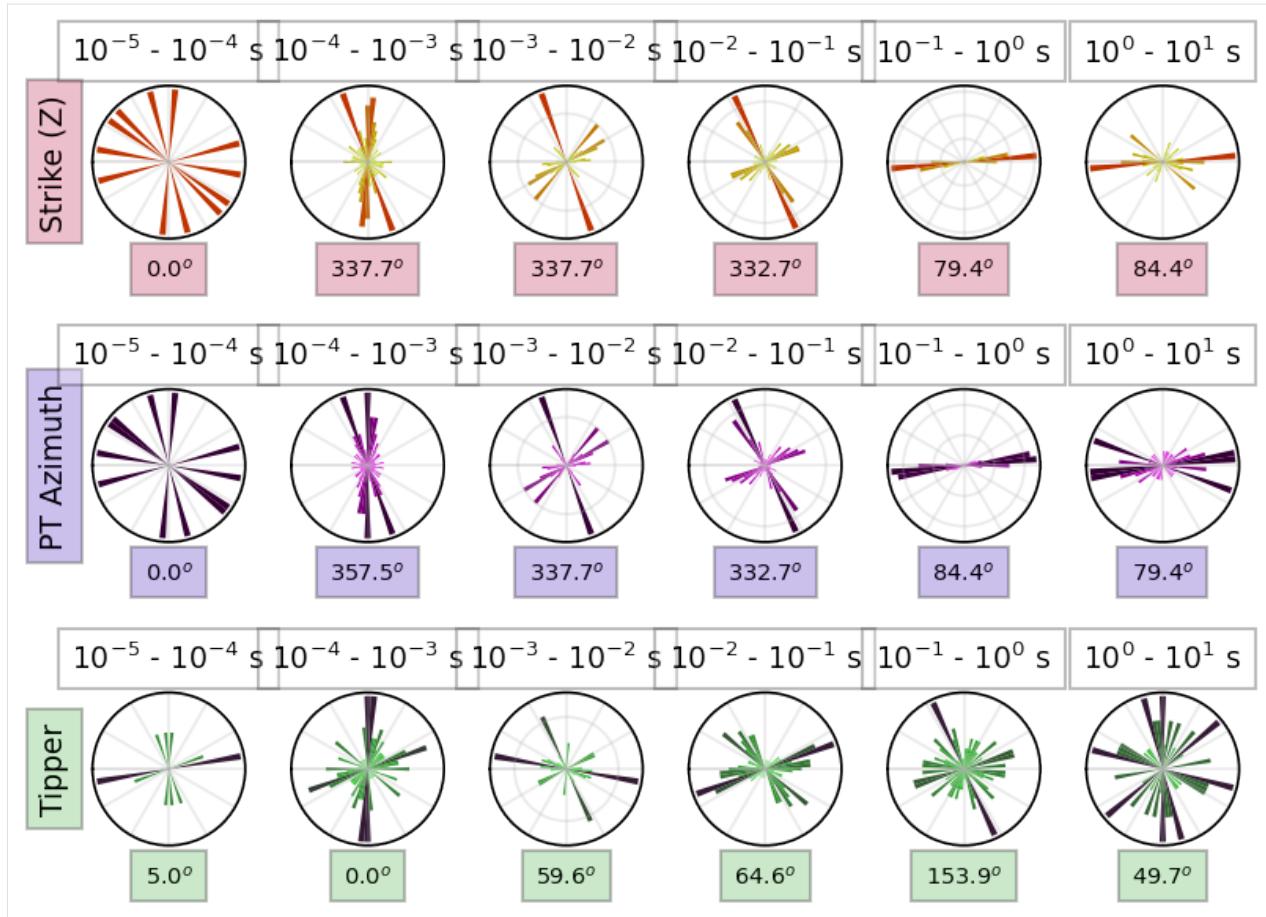


5b. Per Decade

It can be informative to plot the strike angles per decade in period. This can provide information on if strike angle changes with depth. Because this is AMT data there isn't really a coherent strike angle until about 0.1 seconds. Notice the induction vector (tipper) strike is perpendicular to the impedance strike below 0.1 seconds.

```
[14]: strike_plot_per_decade = mtd.plot_strike(plot_type=1, print_stats=True)

Strike statistics for invariant period range 1e-05 to 0.0001 (s) median=290.5 mode=0.0
↳mean=218.3
Strike statistics for pt period range 1e-05 to 0.0001 (s) median=290.8 mode=0.0 mean=217.
↳8
Strike statistics for tipper period range 1e-05 to 0.0001 (s) median=35.3 mode=5.0
↳mean=32.6
Strike statistics for invariant period range 0.0001 to 0.001 (s) median=286.1 mode=337.7
↳mean=198.0
Strike statistics for pt period range 0.0001 to 0.001 (s) median=285.1 mode=357.5
↳mean=198.5
Strike statistics for tipper period range 0.0001 to 0.001 (s) median=1.3 mode=0.0
↳mean=350.6
Strike statistics for invariant period range 0.001 to 0.01 (s) median=76.4 mode=337.7
↳mean=156.6
Strike statistics for pt period range 0.001 to 0.01 (s) median=76.4 mode=337.7 mean=156.8
Strike statistics for tipper period range 0.001 to 0.01 (s) median=334.4 mode=59.6
↳mean=354.5
Strike statistics for invariant period range 0.01 to 0.1 (s) median=291.6 mode=332.7
↳mean=195.9
Strike statistics for pt period range 0.01 to 0.1 (s) median=291.7 mode=332.7 mean=195.8
Strike statistics for tipper period range 0.01 to 0.1 (s) median=307.4 mode=64.6
↳mean=338.6
Strike statistics for invariant period range 0.1 to 1 (s) median=81.5 mode=79.4 mean=93.9
Strike statistics for pt period range 0.1 to 1 (s) median=81.2 mode=84.4 mean=93.7
Strike statistics for tipper period range 0.1 to 1 (s) median=67.1 mode=153.9 mean=37.0
Strike statistics for invariant period range 1 to 10 (s) median=83.5 mode=84.4 mean=146.4
Strike statistics for pt period range 1 to 10 (s) median=79.5 mode=79.4 mean=131.6
Strike statistics for tipper period range 1 to 10 (s) median=357.9 mode=49.7 mean=5.0
23:10:23T09:21:42 | INFO | line:793 |mtpy.imaging.plot_strike | _plot_per_period | Note:
↳North is assumed to be 0 and the strike angle is measured clockwise positive.
```



6. Rotate

Rotation is common for modeling especially 2D because we want the TE mode to be parallel to the profile line and TM to be perpendicular. The strike plots above are good at indicating the dominant strike direction. From above the dominant strike direction is around N85E. Therefore if we want to rotate the data to a profile parallel with geoelectric strike we rotate by N85W or -85 degrees.

```
[15]: mtd.rotate(-85)

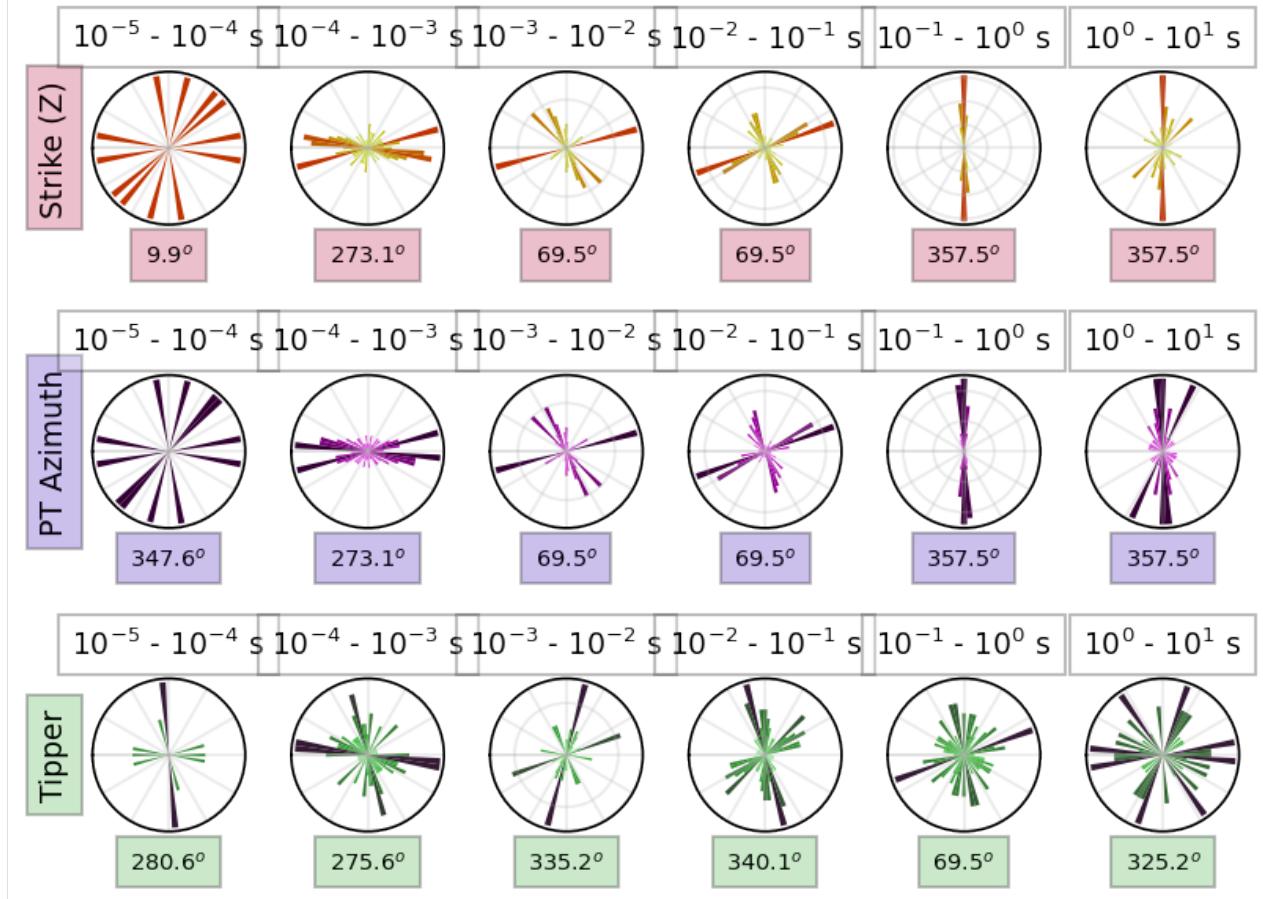
23:10:23T09:22:40 | INFO | line:131 |mtpy.core.mt | rotate | Rotated transfer function
˓→ by: -85.000 degrees clockwise
23:10:23T09:22:40 | INFO | line:131 |mtpy.core.mt | rotate | Rotated transfer function
˓→ by: -85.000 degrees clockwise
23:10:23T09:22:40 | INFO | line:131 |mtpy.core.mt | rotate | Rotated transfer function
˓→ by: -85.000 degrees clockwise
23:10:23T09:22:40 | INFO | line:131 |mtpy.core.mt | rotate | Rotated transfer function
˓→ by: -85.000 degrees clockwise
23:10:23T09:22:40 | INFO | line:131 |mtpy.core.mt | rotate | Rotated transfer function
˓→ by: -85.000 degrees clockwise
23:10:23T09:22:41 | INFO | line:131 |mtpy.core.mt | rotate | Rotated transfer function
˓→ by: -85.000 degrees clockwise
```

6a. Plot Strike

Now if we plot the strike we see that the dominant strike is near 0. Of course you can tweek the rotation angle to get to 0, but this is close enough for demonstration purposes.

```
[16]: rotated_strike = mtd.plot_strike(plot_type=1)
```

23:10:23T09:22:43 | INFO | line:793 |mtpy.imaging.plot_strike | _plot_per_period | Note:
→ North is assumed to be 0 and the strike angle is measured clockwise positive.



7. Interpolate

If you have different transfer functions processed slightly differently then you'll likely have a mismatch in period and for modeling or plotting purposes its nice to have a single period index for all transfer functions.

You can identify the min and max from the data, if that's what you want and then create period from there. From above it looks like the period range is from 10^{-4} to 0.5 seconds. Here we will interpolate onto 23 periods in that range.

```
[17]: import numpy as np
```

```
[18]: new_periods = np.logspace(-4, np.log10(0.5), 23)
```

```
[19]: interpolated_mtd = mtd.interpolate(new_periods, inplace=False)
```

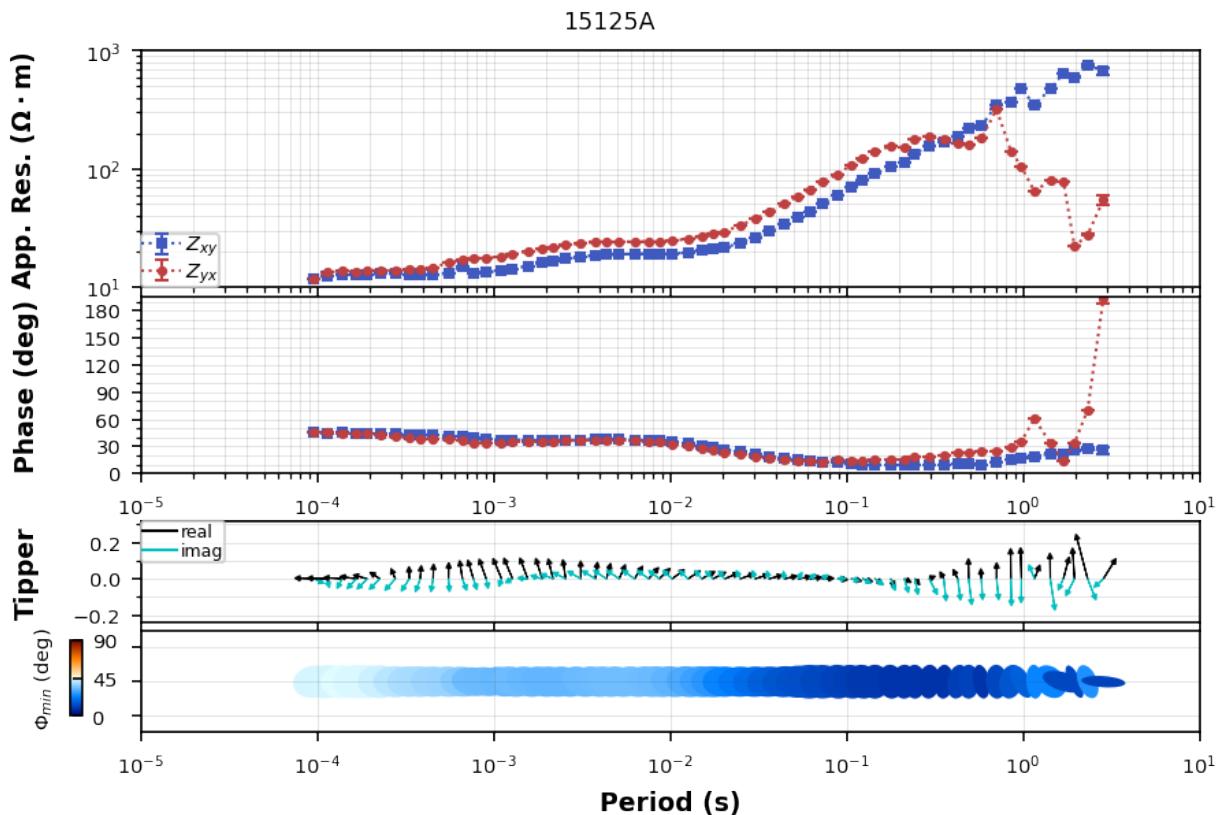
```
[20]: interpolated_mtd
```

```
[20]: MTData([('profile.15125A',
    TF( survey='profile', station='15125A', latitude=-22.37, longitude=149.19,
    ↵ elevation=200.00 )),
    ('profile.15126A',
    TF( survey='profile', station='15126A', latitude=-22.37, longitude=149.19,
    ↵ elevation=200.00 )),
    ('profile.15127A',
    TF( survey='profile', station='15127A', latitude=-22.37, longitude=149.20,
    ↵ elevation=201.00 )),
    ('profile.15128A',
    TF( survey='profile', station='15128A', latitude=-22.37, longitude=149.20,
    ↵ elevation=200.00 )),
    ('profile.15129A',
    TF( survey='profile', station='15129A', latitude=-22.37, longitude=149.21,
    ↵ elevation=202.00 )),
    ('profile.15130A',
    TF( survey='profile', station='15130A', latitude=-22.37, longitude=149.21,
    ↵ elevation=201.00 ))])
```

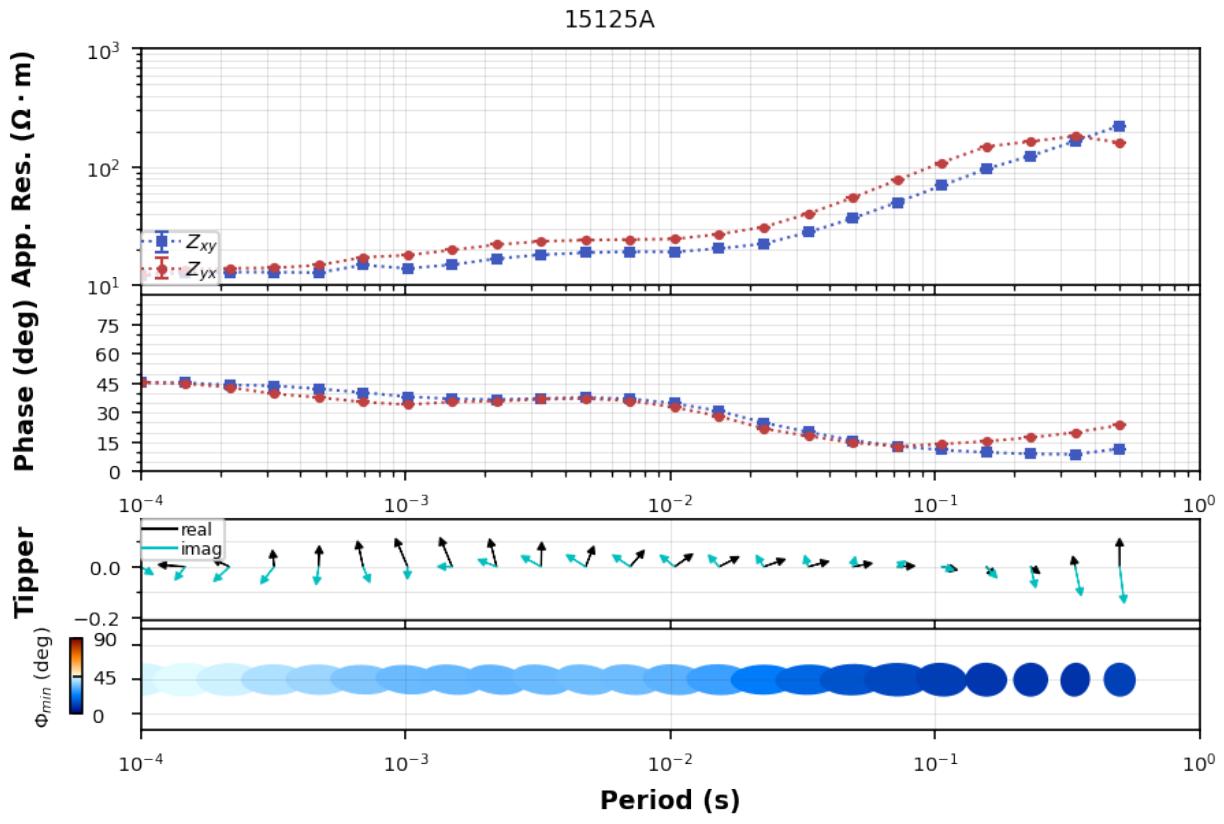
7a. Compare plots

Now we can compare to see how the interpolated transfer function compares to the original. Can plot them individually.

```
[21]: original = mtd.plot_mt_response("profile.15125A")
```



```
[22]: interpolated = interpolated_mtd.plot_mt_response("profile.15125A")
```



7b. Compare in same plot

To plot these in the same plot we need to do some manipulating. First change the survey name in the interpolated data. Then create a new MTData object that is just two transfer functions with the same name but from the original and interpolated data sets.

```
[23]: from mtpy.core.mt_data import MTData
from mtpy.imaging import PlotMultipleResponses
```

```
[24]: new_interpolated_mtd = MTData()
for mt_object in interpolated_mtd.values():
    mt_object.survey_metadata.id = "interpolated"
    new_interpolated_mtd.add_station(mt_object)
```

```
[25]: new_interpolated_mtd.station_locations
```

	survey	station	latitude	longitude	elevation	datum_epsg	\
0	interpolated	15125A	-22.370806	149.188639	200.0	4326	
1	interpolated	15126A	-22.370639	149.193500	200.0	4326	
2	interpolated	15127A	-22.371028	149.198417	201.0	4326	
3	interpolated	15128A	-22.370861	149.203306	200.0	4326	
4	interpolated	15129A	-22.371083	149.208083	202.0	4326	
5	interpolated	15130A	-22.371222	149.212972	201.0	4326	

(continues on next page)

(continued from previous page)

	east	north	utm_epsg	model_east	model_north	\
0	725360.330833	7.524490e+06	32755	-1252.884265	35.819931	
1	725861.314778	7.524502e+06	32755	-751.900320	46.986774	
2	726367.124612	7.524451e+06	32755	-246.090486	-3.473507	
3	726870.972550	7.524462e+06	32755	257.757452	7.618863	
4	727362.745811	7.524430e+06	32755	749.530712	-24.206492	
5	727866.099363	7.524408e+06	32755	1252.884265	-46.986774	
	model_elevation	profile_offset				
0	200.0	0.0				
1	200.0	0.0				
2	201.0	0.0				
3	200.0	0.0				
4	202.0	0.0				
5	201.0	0.0				

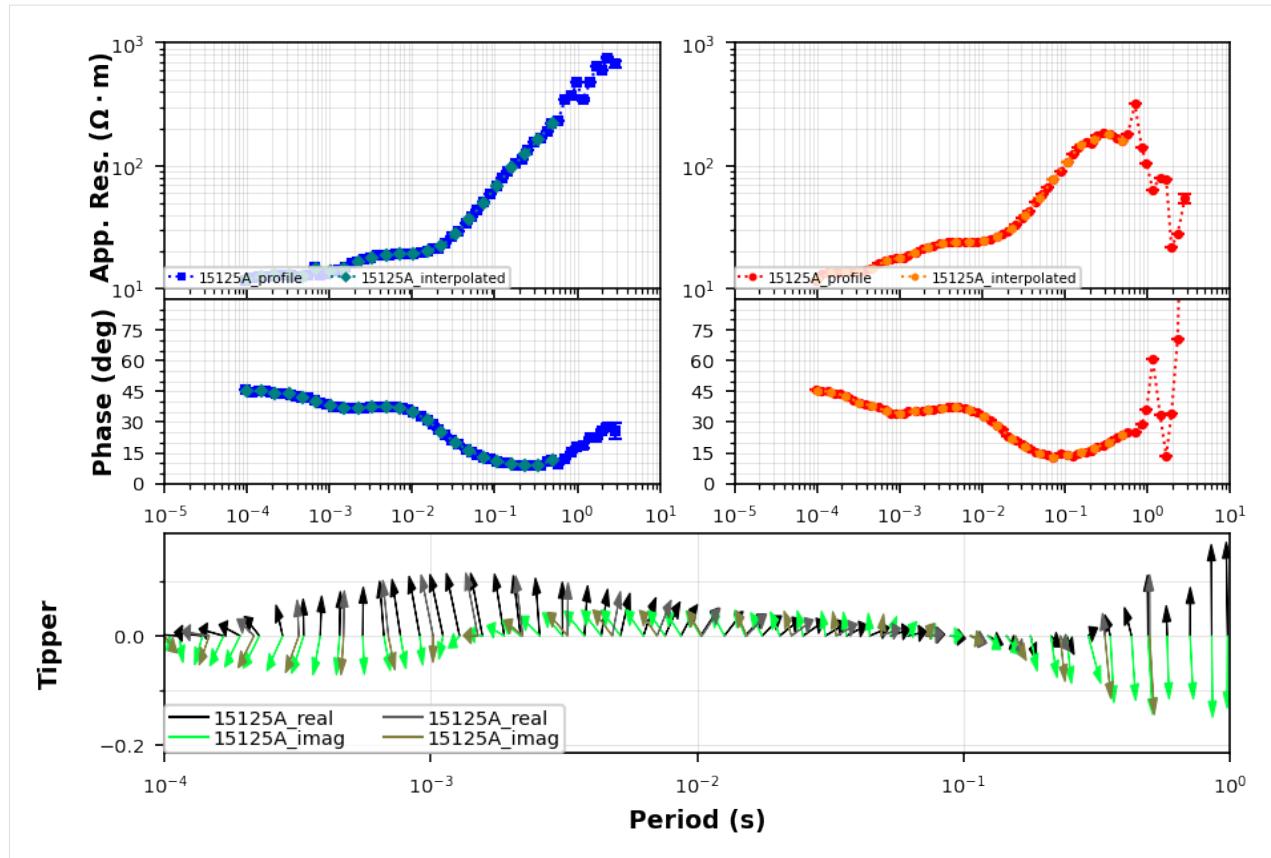
[26]:

```
compare_mtd = MTData()
compare_mtd.add_station(mtd.get_station("15125A", "profile"))
compare_mtd.add_station(new_interpolated_mtd.get_station("15125A", "interpolated"))
```

[27]:

```
compare_same_plot = PlotMultipleResponses(
    compare_mtd, plot_style="compare", plot_tipper="yri", fig_num=4
)
```

<Figure size 432x288 with 0 Axes>



8. Occam 2D

Occam2D [deGroot-Hedlin and Constable \(1990\)](#) is a classic 2D inversion program. To use it you will have to compile it on your machine. For details see <https://marineemlab.ucsd.edu/Projects/Occam/index.html>.

Here we can create input files for Occam2D. Our data is already in an E-W profile and geoelectric strike suggests that's a relatively good start. The dominant strike appears to be around N30W.

To get an accurate model of the data we want the profile line and the dominant geoelectric strike of the data to be perpendicular. To achieve that you can either project the stations onto a profile perpendicular to the geoelectric strike, or rotate the station data to be perpendicular to a map profile. Here, we will project the stations onto a profile perpendicular to geoelectric strike.

8a. Project stations to Geoelectric Strike

When using geoelectric strike to project stations, this is saying that the stations should project onto a profile line that is perpendicular to geoelectric strike such that when you model the data the TE mode (electric along strike) is perpendicular to the profile and the TM mode (electric perpendicular to strike) is parallel to the profile. You can read much more complete explanations looking up MT papers from the 90's. [Wannamaker et al. \(1984\)](#) is a good start.

```
[28]: x1, y1, x2, y2, strike_profile = interpolated_mtd.generate_profile_from_strike(-30)
strike_profile
```

```
[28]: {'slope': 1.1253388328842984, 'intercept': -190.2589909890415}
```

```
[29]: geoelectric_strike_mtd = interpolated_mtd.get_profile(x1, y1, x2, y2, 5000)
```

```
[30]: geoelectric_strike_mtd
[30]: MTData([('interpolated.15125A',
              TF( survey='interpolated', station='15125A', latitude=-22.37, longitude=149.19,
              ↪elevation=200.00 )),
             ('interpolated.15126A',
              TF( survey='interpolated', station='15126A', latitude=-22.37, longitude=149.19,
              ↪elevation=200.00 )),
             ('interpolated.15127A',
              TF( survey='interpolated', station='15127A', latitude=-22.37, longitude=149.20,
              ↪elevation=201.00 )),
             ('interpolated.15128A',
              TF( survey='interpolated', station='15128A', latitude=-22.37, longitude=149.20,
              ↪elevation=200.00 )),
             ('interpolated.15129A',
              TF( survey='interpolated', station='15129A', latitude=-22.37, longitude=149.21,
              ↪elevation=202.00 )),
             ('interpolated.15130A',
              TF( survey='interpolated', station='15130A', latitude=-22.37, longitude=149.21,
              ↪elevation=201.00 ))])
```

8b. Compute Model Errors

For 2D modeling its usually more advantageous to invert the apparent resistivity and phase of the TE and TM modes. We can set the error for each.

```
[31]: occam2d_object = geoelectric_strike_mtd.to_occam2d_data(geoelectric_strike=-30, profile_
   ↪angle=60)
```

Set resistivity error

Its common to give the resistivity components a more uncertainty because of static shifts. Here we will give each component a 20% absolute error and set it to a logarithmic representation.

```
[38]: occam2d_object.dataframe["res_xy_model_error"] = (20 / 100) / np.log(10.)
occam2d_object.dataframe["res_yx_model_error"] = (20 / 100) / np.log(10.)
```

Set Phase error

The phase is insensitive to near surface heterogeneties and can have smaller uncertainties. Here we give 2.5%.

```
[43]: occam2d_object.dataframe["phase_xy_model_error"] = (5 / 100.) * 57. / 2.
occam2d_object.dataframe["phase_yx_model_error"] = (2.5 / 100.) * 57. / 2.
```

Write the data file

Now we can write the data file and we should pick which components to invert for. By looking at the help we can see which combinations are supported. We will use mode 4 to invert both TE and TM modes in log space.

```
[44]: help(occam2d_object)
```

```
Help on Occam2DDData in module mtpy.modeling.occam2d.data object:
```

```
class Occam2DDData(builtins.object)
|   Occam2DDData(dataframe=None, center_point=None, **kwargs)
|
|   Reads and writes data files and more.
|
|   Inherits Profile, so the intended use is to use Data to project stations
|   onto a profile, then write the data file.
|
|   =====
|   Model Modes          Description
|   =====
|   1 or log_all         Log resistivity of TE and TM plus Tipper
|   2 or log_te_tip      Log resistivity of TE plus Tipper
|   3 or log_tm_tip      Log resistivity of TM plus Tipper
|   4 or log_te_tm       Log resistivity of TE and TM
|   5 or log_te          Log resistivity of TE
|   6 or log_tm          Log resistivity of TM
|   7 or all             TE, TM and Tipper
|   8 or te_tip          TE plus Tipper
|   9 or tm_tip          TM plus Tipper
|   10 or te_tm          TE and TM mode
|   11 or te             TE mode
|   12 or tm             TM mode
|   13 or tip            Only Tipper
|   =====
|
|   :Example Write Data File: ::

|       >>> from mtpy.modeling.occam2d import Data
|       >>> occam_data_object = Data()
|       >>> occam_data_object.read_data_file(r"path/to/data/file.dat")
|       >>> occam_data_object.model_mode = 2
|       >>> occam_data_object.write_data_file(r"path/to/new/data/file_te.dat")

Methods defined here:

__init__(self, dataframe=None, center_point=None, **kwargs)
    Initialize self. See help(type(self)) for accurate signature.

__repr__(self)
    Return repr(self).

__str__(self)
    Return str(self).

mask_from_datafile(self, mask_datafn)
    reads a separate data file and applies mask from this data file.
    mask_datafn needs to have exactly the same frequencies, and station names
    must match exactly.
```

(continues on next page)

(continued from previous page)

```

| read_data_file(self, data_fn=None)
|     Read in an existing data file and populate appropriate attributes
|         * data
|         * data_list
|         * freq
|         * station_list
|         * station_locations
|
| Arguments:
| -----
|     **data_fn** : string
|             full path to data file
|             *default* is None and set to save_path/fn_basename
|
| :Example: ::

|     >>> import mtpy.modeling.occam2d as occam2d
|     >>> ocd = occam2d.Data()
|     >>> ocd.read_data_file(r"/home/Occam2D/Line1/Inv1/Data.dat")

| write_data_file(self, data_fn=None)
|     Write a data file.

| Arguments:
| -----
|     **data_fn** : string
|             full path to data file.
|             *default* is save_path/fn_basename

| If there data is None, then _fill_data is called to create a profile,
| rotate data and get all the necessary data. This way you can use
| write_data_file directly without going through the steps of projecting
| the stations, etc.

| :Example: ::

|     >>> edipath = r"/home/mt/edi_files"
|     >>> slst = ['mt{0:03}'.format(ss) for ss in range(1, 20)]
|     >>> ocd = occam2d.Data(edi_path=edipath, station_list=slst)
|     >>> ocd.save_path = r"/home/occam/line1/inv1"
|     >>> ocd.write_data_file()

| -----
| Readonly properties defined here:

frequencies
n_data
n_frequencies
n_stations

```

(continues on next page)

(continued from previous page)

```
| offsets
| stations
|
| -----
| Data descriptors defined here:
|
| __dict__
|     dictionary for instance variables (if defined)
|
| __weakref__
|     list of weak references to the object (if defined)
|
| data_filename
|
| dataframe
```

```
[45]: occam2d_object.model_mode = "4"
```

```
[46]: occam2d_object.write_data_file(Path().joinpath("occam2d_example.dat"))
```

1.2.4 MTData: Grid Example

It is becoming more common for MT data to be collected in a grid rather than a profile because of 3D inversions. In this example we can take a look at how to work with MT data collected on a grid. We will use the same MTCollection from the first example.

1. Load data

First we will open the MTCollection and change the working dataframe to get only those station in the ‘grid’ survey.

```
[1]: from pathlib import Path
from mtpy import MTCollection
```

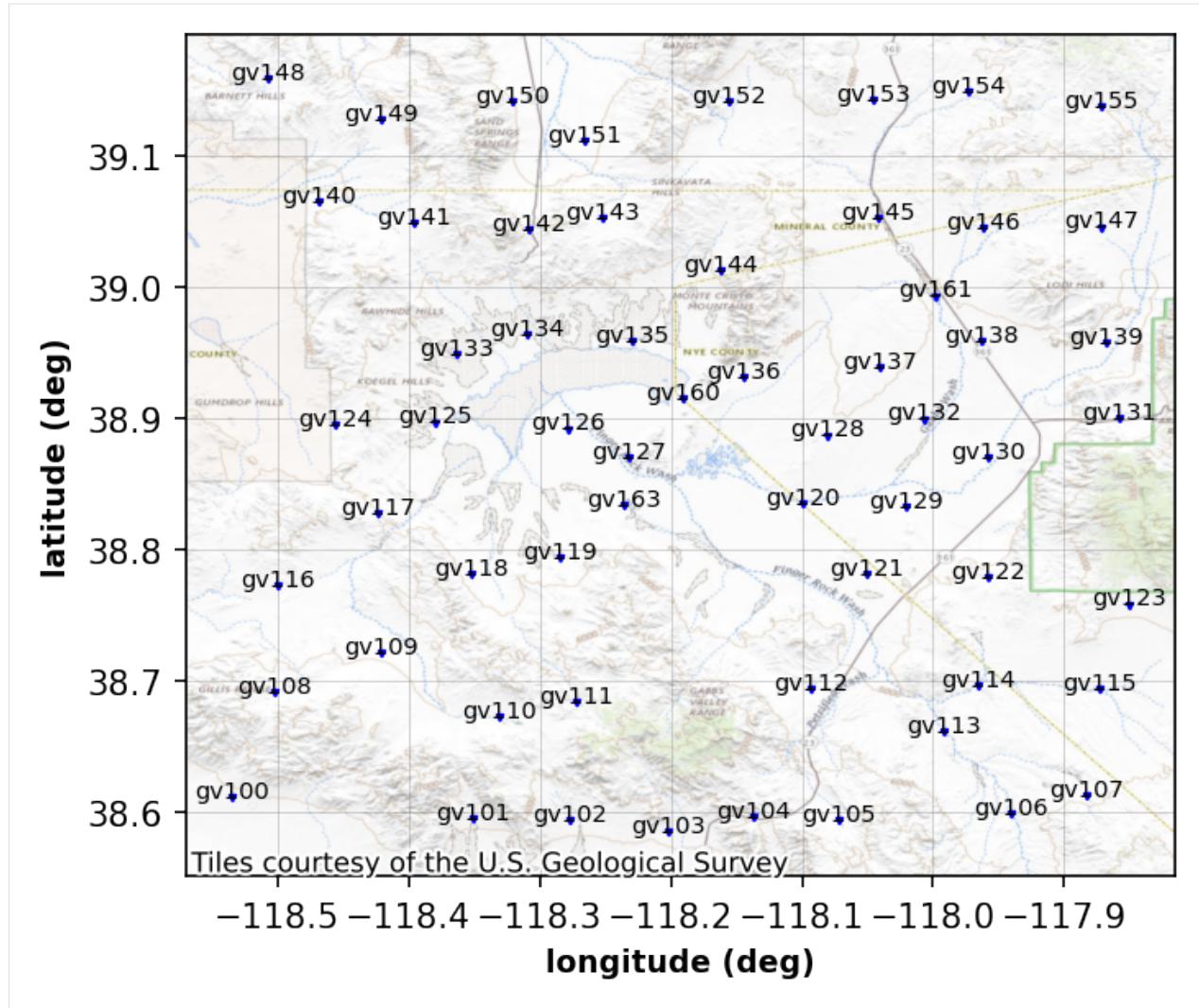
```
[2]: with MTCollection() as mtc:
    mtc.open_collection(Path().cwd().joinpath("test_mt_collection.h5"))
    mtc.working_dataframe = mtc.master_dataframe.loc[mtc.master_dataframe.survey == "grid"]
    mtd = mtc.to_mt_data()

23:10:23T09:44:34 | INFO | line:760 |mth5.mth5 | close_mth5 | Flushing and closing C:\Users\jpeacock\OneDrive - DOI\Documents\GitHub\mtpy-v2\docs\source\notebooks\test_mt_collection.h5
```

2. Plot Station Locations

To make sure we got the data we expected, we can plot the station locations. You can change the basemap, see [providers](#) for more details

```
[3]: station_plot = mtd.plot_stations()
```

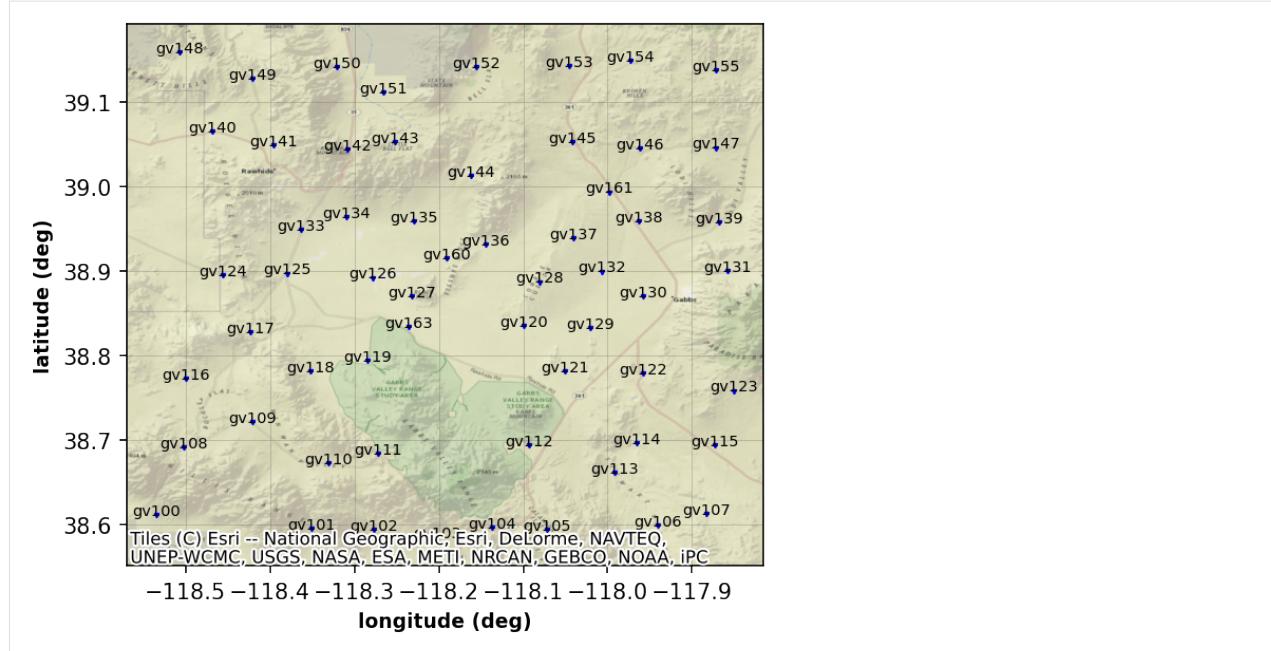


2a. Change basemap

Here is an example of how to change the basemap. We will use the ESRI terrain map.

```
[4]: import contextily as cx
```

```
[5]: station_plot.cx_source = cx.providers.Esri.NatGeoWorldMap
station_plot.redraw_plot()
```

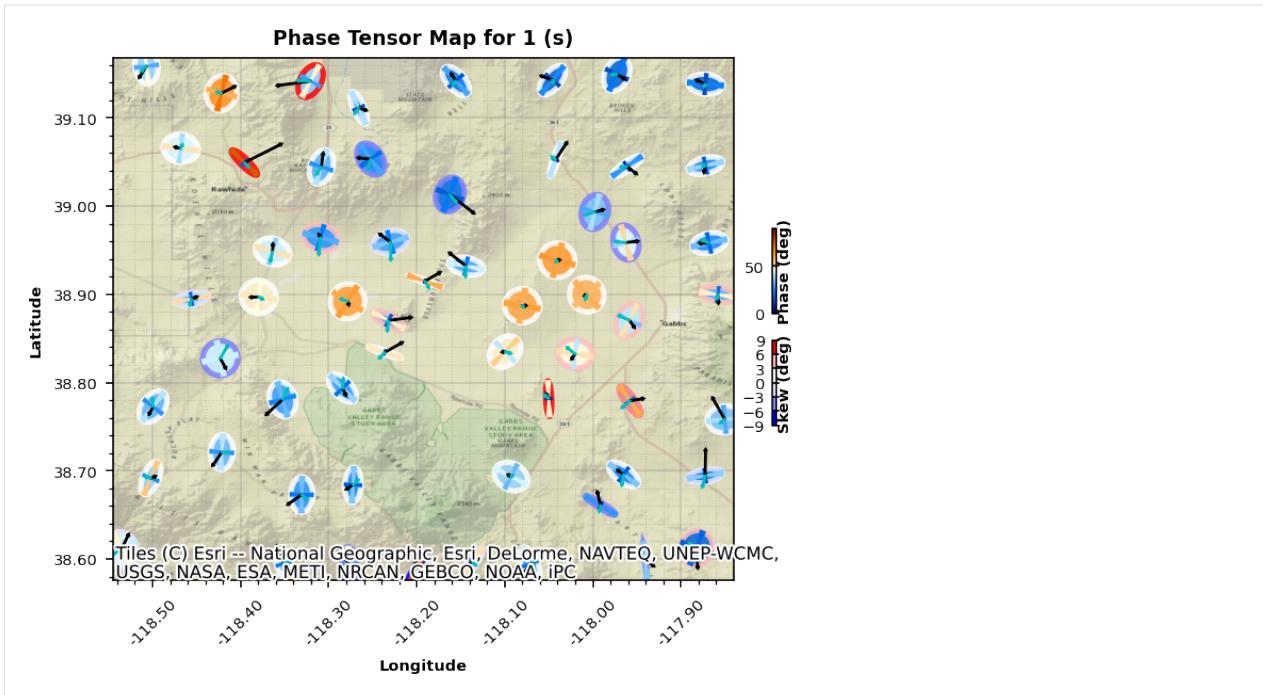


3. Plot Phase Tensor Map

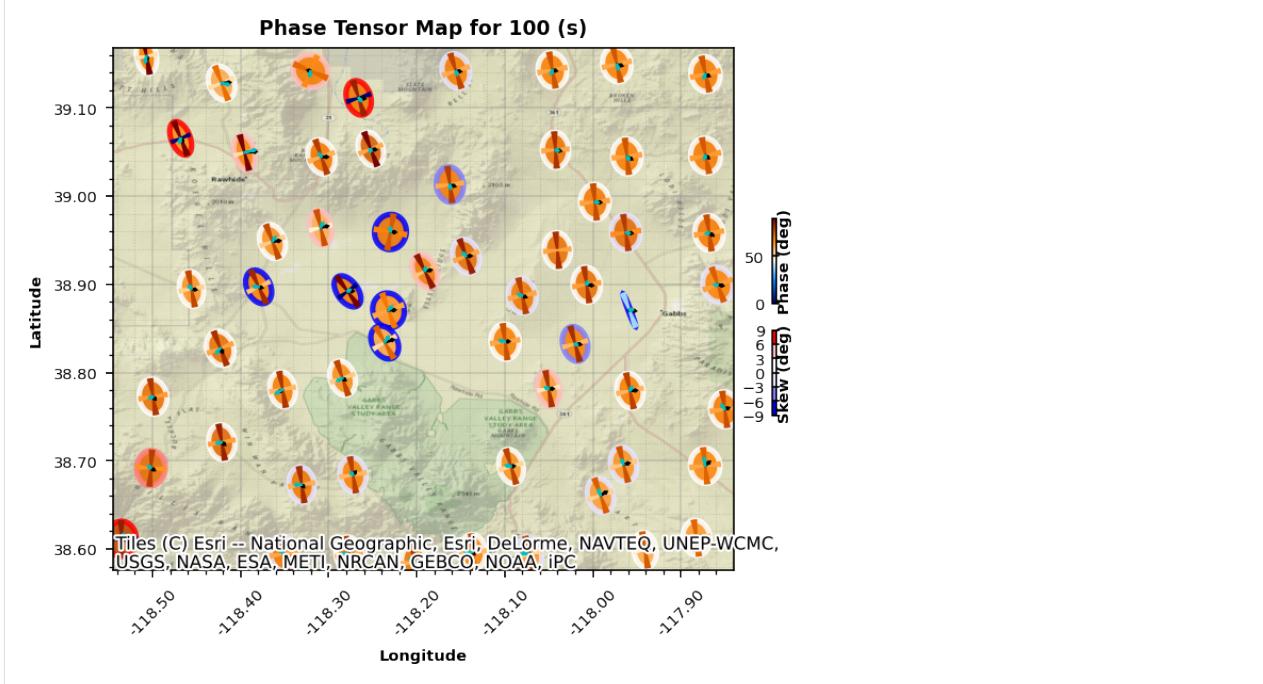
Now that we seem to have the correct data, lets plot phase tensor maps. These are broadband data with induction vectors, so lets plot those too. The phase tensor map is busy so lets identify key aspects.

- Ellipse shape: the ellipses often elongate in the preferred direction of current flow
- Ellipse face color: by default are colored by the parameter `phimin` but can be changed by setting the attribute `pt_map.ellipse_colorby`. The `phimin` gives the lower bounds on how the subsurface resistivity is changing, where reds are becoming more conductive and blues are becoming more resistive.
- Ellipse edge color: by default is colored by the skew angle which is indicative of dimensionality with high skew being 3D. Also the color indicates in which direction currents are being skewed.
- Wedges: the long axis is `phimax` and the short axis are `phimin`.
- Arrows: black are real induction vectors and blue are imaginary induction vectors.

```
[6]: pt_map = mtd.plot_phase_tensor_map(
    plot_tipper="yri",
    cx_source=cx.providers.Esri.NatGeoWorldMap,
    ellipse_size=.02,
    arrow_size=.05
)
```



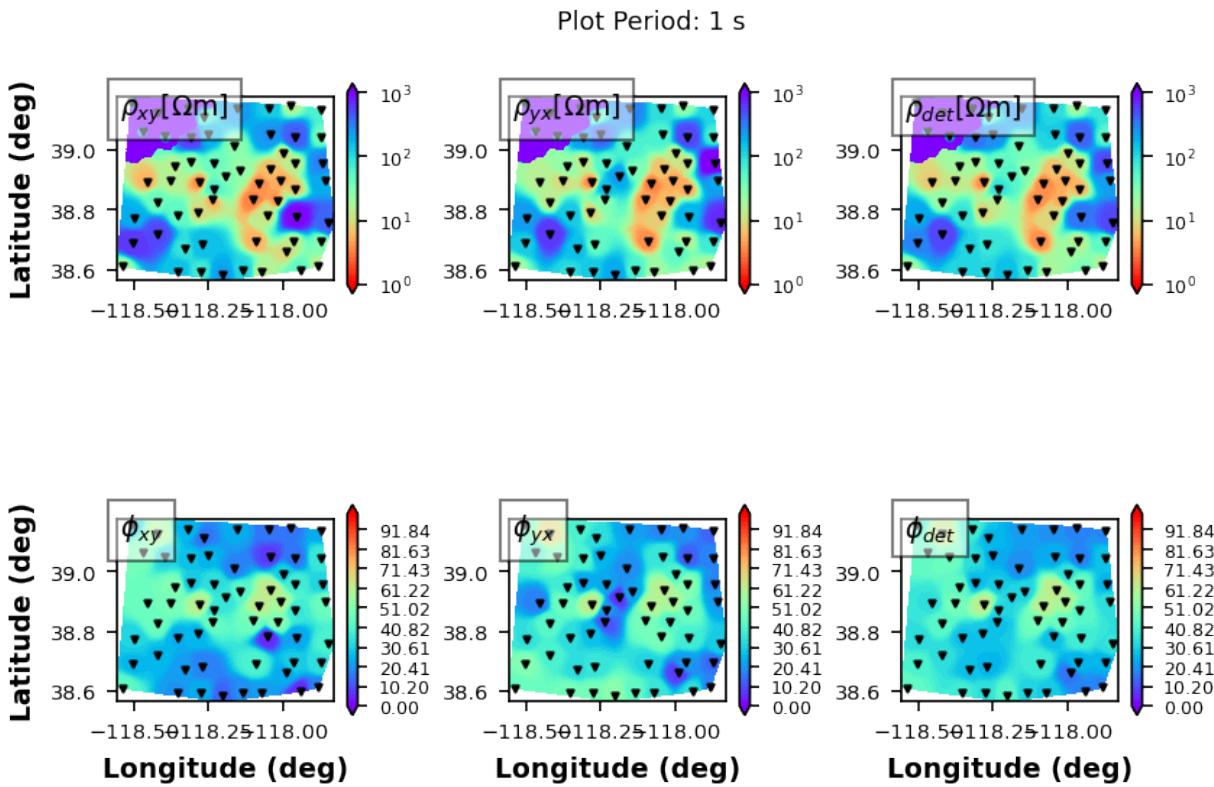
```
[7]: pt_map.plot_period = 100
pt_map.redraw_plot()
```



4. Plot Resistivity and Phase Maps

An informative first order check on the data is to plot apparent resistivity and phase maps of the data. This can help identify odd stations for further investigations and get a general idea of subsurface resistivity changes. These plots are a little messy in a Jupyter Notebook.

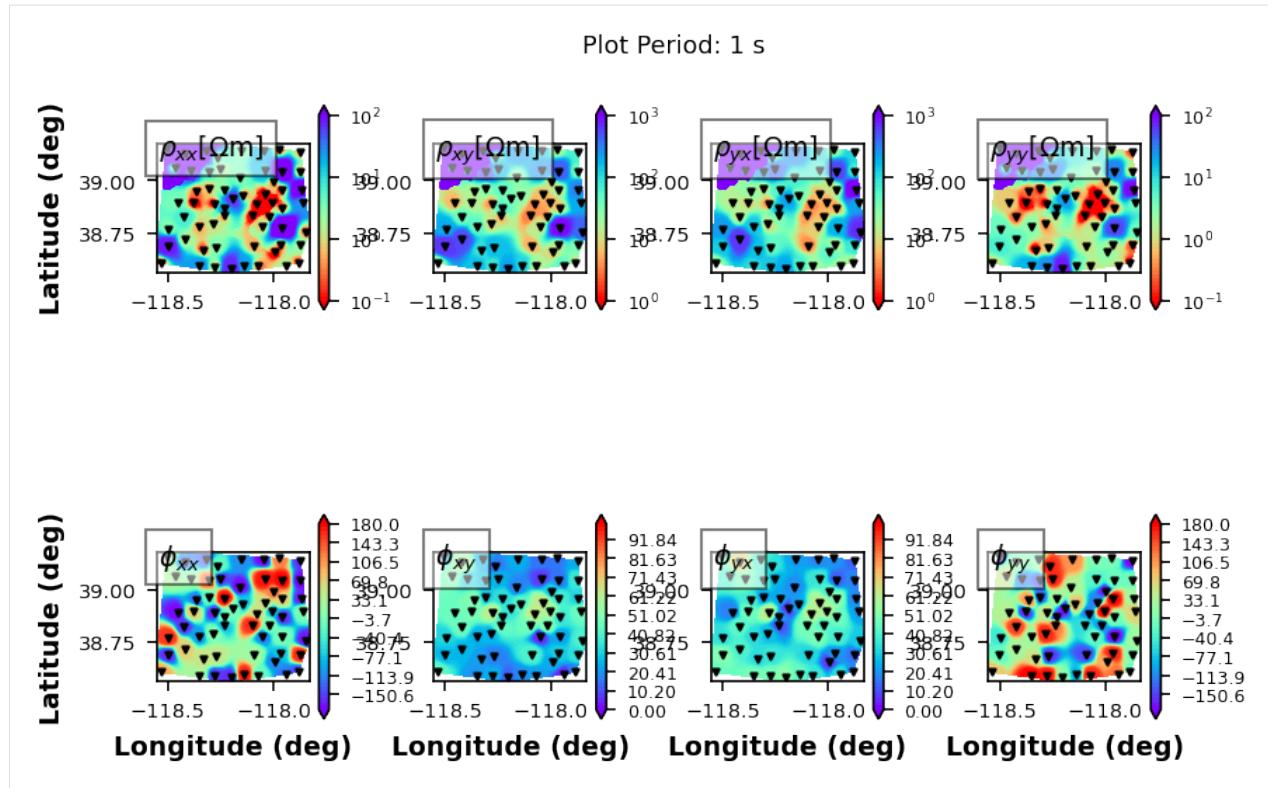
```
[8]: rp_map = mtd.plot_resistivity_phase_maps(plot_det=True, subplot_wspace=.45, marker_size=4)
```



4a. Plot Diagonal Components

It can be informative to plot the diagonal components as well, in this case they outline the basins nicely.

```
[9]: rp_map.plot_xx = True
rp_map.plot_yy = True
rp_map.plot_det = False
rp_map.redraw_plot()
```

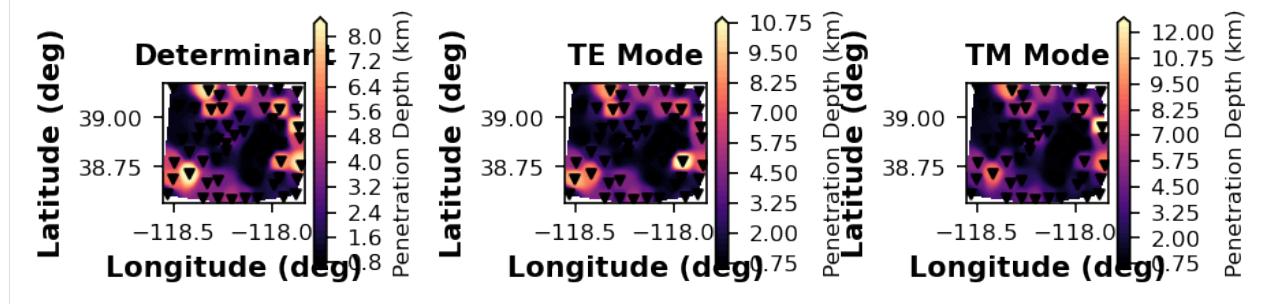


5. Plot Depth of Investigation

It can also be informative to understand how deep the measurements are sensitive to. Here a Niblett-Bostick approximation is used to estimate the depth of penetration for each station at a single period.

```
[10]: depth_of_penetration = mtd.plot_penetration_depth_map(subplot_wspace=.35)
```

Depth of investigation for period 1 (s)



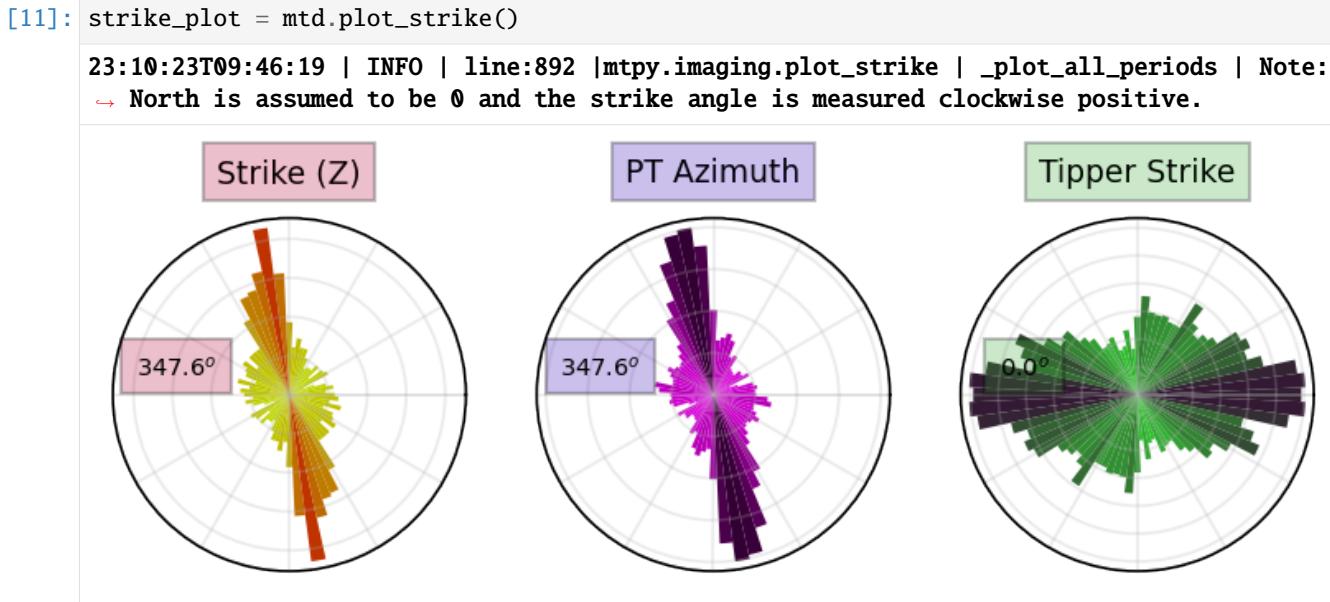
6. Plot Strike

Here we will plot all periods of estimated strike. Notice that the plot includes the strike as estimated from the invariants (left) of Weaver et al. (2002), the phase tensor (middle) of Caldwell et al. (2004), and the induction vector strike.

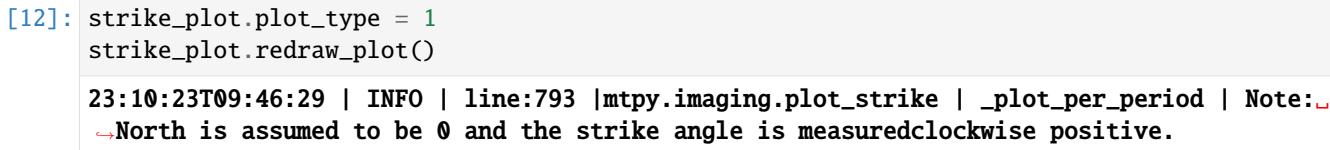
Important: The induction strike points towards good conductors so should therefore be perpendicular to the impedance strike. We left it this way as a sanity check on strike angles.

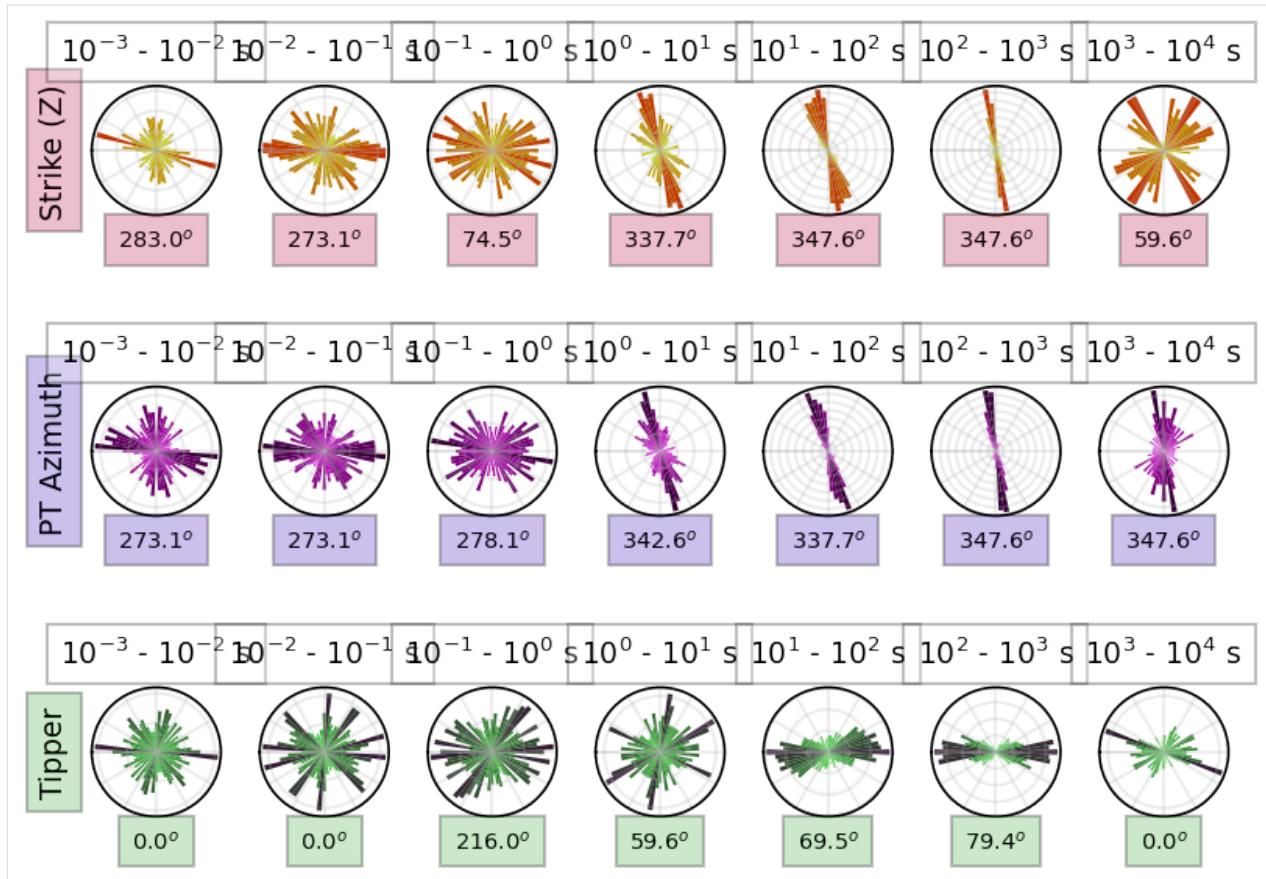
6a. Plot all periods in one plot

This plot shows all strike estimations for all stations for all periods.



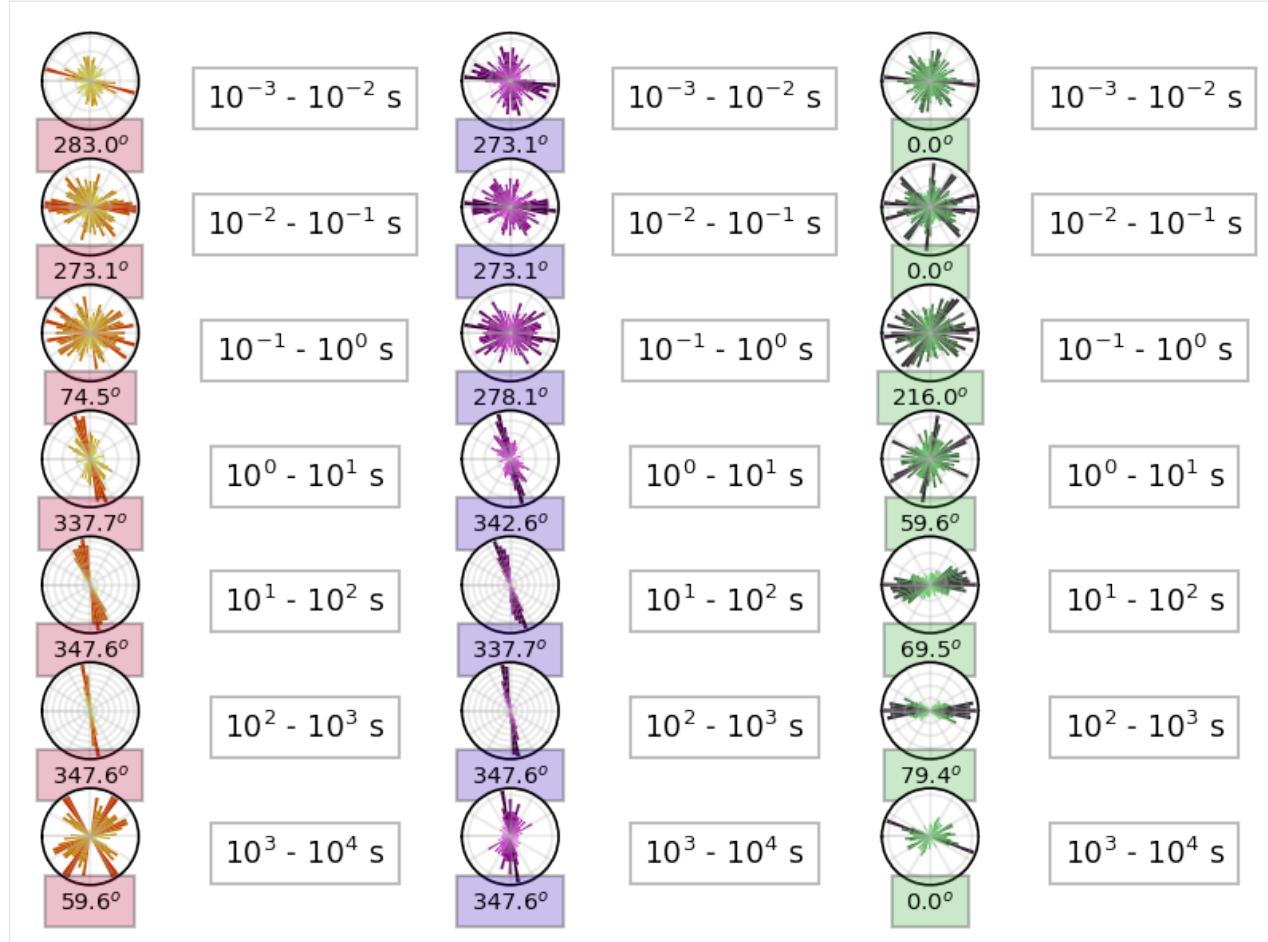
6b. Strike per period





```
[13]: strike_plot.plot_orientation = "vertical"
strike_plot.redraw_plot()
```

```
23:10:23T09:47:25 | INFO | line:793 |mtpy.imaging.plot_strike | _plot_per_period | Note:
--North is assumed to be 0 and the strike angle is measured clockwise positive.
```



7. Extract a Profile

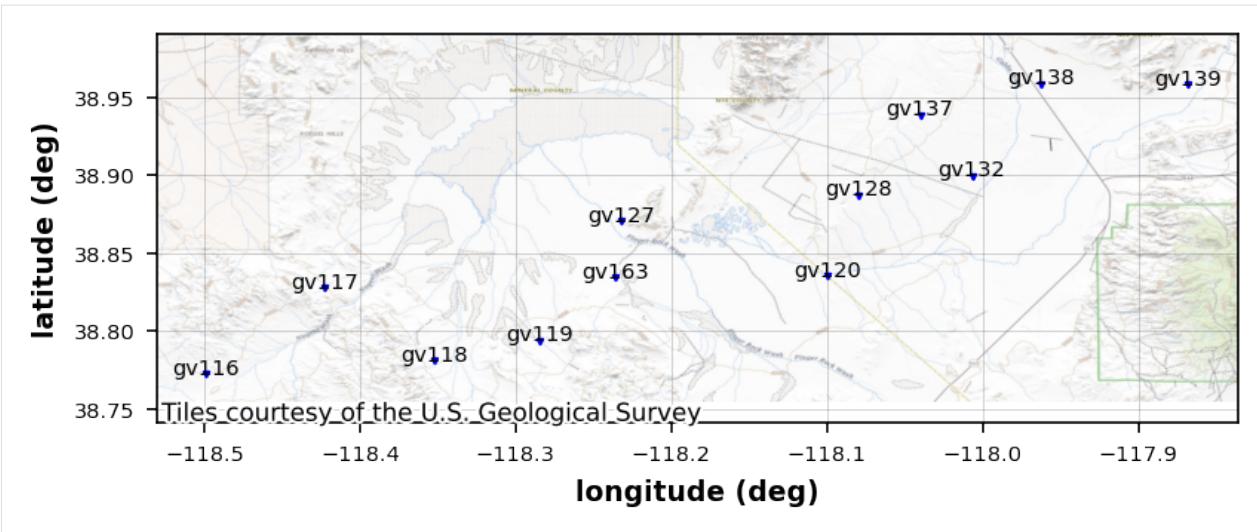
One advantageous way of looking at grid data is to extract profiles across interesting structures. This can be done by knowing the start and end points of the profile you would like to look at and providing a distance away from the profile line to include.

By analyzing the strike direction we can see that a profile approximately N75E would be perpendicular to geoelectric strike. Lets cut across the basins.

```
[17]: mtd.utm_crs = 32611
```

```
[22]: %time
profile = mtd.get_profile(-118.45, 38.78, -117.85, 38.95, 5000)
Wall time: 2min 48s
```

```
[23]: profile_stations_plot = profile.plot_stations()
```



[]:

1.2.5 Make 3D Data Files

Inversion programs need input data, MTpy has tools for creating model files for various modeling programs. This is an example for creating a data file for ModEM using the example grid data.

Imports

[1]: %matplotlib widget

[2]:

```
from pathlib import Path
from mtpy import MTCollection
```

1. Load in Data

Load in the data created in the first example and get only the data collected on a grid. Turn those data into a MTData object from which we can manipulate the data.

Tip: Using the `with` context manager will close the MTH5 when finished. This is safer because if something goes wrong the file will always be closed and reduce the chance of corruption.

[3]:

```
with MTCollection() as mtc:
    mtc.open_collection(Path().cwd().joinpath("test_mt_collection.h5"))
    mtc.working_dataframe = mtc.master_dataframe.loc[mtc.master_dataframe.survey == "grid"]
    mtd = mtc.to_mt_data()
```

23:10:20T10:27:58 | INFO | line:760 |mth5.mth5 | close_mth5 | Flushing and closing C:\Users\jpeacock\OneDrive - DOI\Documents\GitHub\mtpy-v2\docs\source\notebooks\test_mt_collection.h5

2. Calculate Relative Locations

Most modeling programs are agnostic to geographic coordinates as they use a relative coordinate system usually with (0, 0, 0) at one of the corners or the center of the mesh. Here we assume the center of the station area is the (0, 0) point and relative locations are computed relative to the center point. All relative locations are in meters.

Important: To calculate relative locations you must set the ‘utm_epsg’ or ‘utm_crs’ if you have a custom datum. Once you set the ‘utm_crs’ or ‘utm_epsg’ easting and northing is estimated for each station in the MTData object. This can take a few seconds if there are a lot of stations.

Here we will set the EPSG number to WGS84 UTM grid 11N (32611).

Tip: To access station locations of the MTData object use MTData.station_locations. This returns a Pandas DataFrame which can be easily manipulated.

Tip: The center point is MTData.center_point and is estimated from the station locations in the MTData object. You can set the center latitude and longitude if you want to offset the center.

```
MTData._center_lat = new_center_latitude  
MTData._center_lon = new_center_longitude  
MTData._center_elev = new_center_elevation
```

```
[4]: mtd.center_point
```

```
[4]: MT Location:
```

```
-----  
Latitude (deg): 38.871946  
Longitude (deg): -118.192737  
Elevation (m): 0.0000  
Datum crs: epsg:4326  
  
Easting (m): 0.000  
Northing (m): 0.000  
UTM crs: None  
  
Model Easting (m): 0.000  
Model Northing (m): 0.000  
Model Elevation (m): 0.000  
Profile Offset (m): 0.000
```

Station Locations

You can have a look at the station locations retrieved from the MTCollection. Notice there are no values for east and north and model_east and model_north yet.

```
[5]: mtd.station_locations.head(5)
```

```
survey station  latitude  longitude  elevation  datum_epsg  east  north  \  
0  grid  gv100  38.611381 -118.535261  1437.40      4326  0.0   0.0  
1  grid  gv101  38.594561 -118.351111  1540.55      4326  0.0   0.0  
2  grid  gv102  38.593692 -118.276822  1554.80      4326  0.0   0.0  
3  grid  gv103  38.585283 -118.202481  1543.90      4326  0.0   0.0  
4  grid  gv104  38.596456 -118.136547  1801.80      4326  0.0   0.0  
  
utm_epsg  model_east  model_north  model_elevation  profile_offset  
0        None       0.0        0.0          1437.40        0.0
```

(continues on next page)

(continued from previous page)

1	None	0.0	0.0	1540.55	0.0
2	None	0.0	0.0	1554.80	0.0
3	None	0.0	0.0	1543.90	0.0
4	None	0.0	0.0	1801.80	0.0

Set UTM EPSG

Here we set the UTM EPSG number, which creates a `pyproj.CRS` object for easier projection between coordinate systems, mainly from degrees to meters. If you created your own CRS you can set the CRS directly, see `pyproj.CRS` for details on creating a custom CRS. Once either the `utm_epsg` or `utm_crs` is set northing and easting for each station in the `MTData` object is calculated and the station locations are updated, which updates the center point.

```
[6]: mtd.utm_epsg = 32611
```

```
[7]: mtd.center_point
```

```
[7]: MT Location:
```

```
-----
Latitude (deg): 38.873786
Longitude (deg): -118.196245
Elevation (m): 0.0000
Datum crs:      epsg:4326

Easting (m):    396229.649
Northing (m):   4303450.537
UTM crs:        epsg:32611

Model Easting (m): 396229.649
Model Northing (m): 4303450.537
Model Elevation (m): 0.000
Profile Offset (m): 0.000
```

```
[8]: mtd.station_locations.head(5)
```

```
survey station  latitude  longitude  elevation datum_epsg      east \
0  grid  gv100  38.611381 -118.535261  1437.40      4326  366331.327569
1  grid  gv101  38.594561 -118.351111  1540.55      4326  382337.730338
2  grid  gv102  38.593692 -118.276822  1554.80      4326  388806.125501
3  grid  gv103  38.585283 -118.202481  1543.90      4326  395268.278608
4  grid  gv104  38.596456 -118.136547  1801.80      4326  401026.349952

      north utm_epsg  model_east  model_north  model_elevation \
0  4.274770e+06    32611       0.0         0.0       1437.40
1  4.272652e+06    32611       0.0         0.0       1540.55
2  4.272463e+06    32611       0.0         0.0       1554.80
3  4.271442e+06    32611       0.0         0.0       1543.90
4  4.272609e+06    32611       0.0         0.0       1801.80

profile_offset
0            0.0
1            0.0
2            0.0
```

(continues on next page)

(continued from previous page)

3	0.0
4	0.0

Calculate Relative Locations

Now that we have easting and northing we can estimate relative locations in model coordinates, which assumes the center of the station area is (0, 0). If you want to offset the center set these values to your desired center location.

```
MTData._center_lat = new_center_latitude
MTData._center_lon = new_center_longitude
MTData._center_elev = new_center_elevation
```

The method to use is `compute_relative_locations()`. It iterates over the stations and removes the center point from the easting and northing.

```
[9]: mtd.compute_relative_locations()
```

Now we have relative locations.

```
[10]: mtd.station_locations.head(5)
```

```
survey station latitude longitude elevation datum_epsg      east \
0   grid   gv100  38.611381 -118.535261    1437.40      4326  366331.327569
1   grid   gv101  38.594561 -118.351111    1540.55      4326  382337.730338
2   grid   gv102  38.593692 -118.276822    1554.80      4326  388806.125501
3   grid   gv103  38.585283 -118.202481    1543.90      4326  395268.278608
4   grid   gv104  38.596456 -118.136547    1801.80      4326  401026.349952

      north utm_epsg      model_east      model_north      model_elevation \
0  4.274770e+06    32611 -29898.321606 -28680.329764        1437.40
1  4.272652e+06    32611 -13891.918837 -30798.871440        1540.55
2  4.272463e+06    32611 -7423.523674 -30987.924124        1554.80
3  4.271442e+06    32611 -961.370567 -32008.380683        1543.90
4  4.272609e+06    32611  4796.700777 -30841.737287        1801.80

profile_offset
0          0.0
1          0.0
2          0.0
3          0.0
4          0.0
```

3. Compute Model Errors

After getting relative locations we can now calculate model errors. These are often larger than measurement error. People have their favorite method of estimating errors and we have tried to accommodate most. If there isn't one here try adding to `mtpy.modeling.errors` or raise an issue. Supported so far are:

Name	Description	Example
geometric_mean	Geometric mean of the off-diagonal components of the impedance tensor, or components of the induction vectors.	$\text{error_value} \cdot \sqrt{(Z_{xy} \cdot Z_{yx})}$
arithmetic_mean	Arithmetic mean of off-diagonal components of the impedance tensor, or components of the induction vectors.	$\text{error_value} \cdot (Z_{xy} + Z_{yx})/2$
row	Error per row of the impedance tensor	$\text{error_value} \cdot Z_{xy}$
median	Median of the off-diagonal components	$\text{error_value} \cdot \text{median}([Z_{xy}, Z_{yx}])$
eigen	Maximum Eigen value of the impedance tensor	$\text{error_value} \times \text{eigen}(Z)$
percent	Percent error per component	$\text{error_value} \cdot Z_{ij}$
absolute	Absolute error	error_value

Setting Impedance Error

MTData has an attribute `z_model_error` that is an `mtpy.modeling.errors.ModelErrors` object. It has some default values, which seem fine for now. But if you want to change any you can use:

```
mtd.z_model_error.error_type = "row"
mtd.z_model_error.error_value = 10    # for 10 percent
mtd.z_model_error.floor = False      # use the calculated value not as a floor for
# measured errors but absolute value.
```

[11]: `mtd.z_model_error`

[11]: Model Errors:

```
-----
        error_type:    geometric_mean
        error_value:   0.05
        floor:         True
        mode:          impedance
```

Setting Tipper Error

Similar to the impedance MTData has an attribute `t_model_error` and is a `mtpy.modeling.errors.ModelErrors` object.

[12]: `mtd.t_model_error`

[12]: Model Errors:

```
-----
        error_type:    absolute
        error_value:   0.02
        floor:         True
        mode:          tipper
```

Calculate Model Errors

Once you set `z_model_error` and `t_model_error` parameters then run `MTData.compute_model_errors()`

Tip: `compute_model_errors` accepts key words that can be used to set `z_model_error` and `t_model_error`

[13]: `mtd.compute_model_errors(z_error_value=7)`

```
[14]: mtd["grid.gv100"].impedance[0:1]
```

```
[14]: <xarray.DataArray 'impedance' (period: 1, output: 2, input: 2)>
array([[[ -86.57589 -714.197j, 1090.806 +2750.944j],
       [-115.5849 +1316.743j, -683.7422 -5020.612j]]])
```

Coordinates:

```
* output  (output) <U2 'ex' 'ey'
* input   (input) <U2 'hx' 'hy'
* period  (period) float64 0.001302
```

Attributes:

```
survey:           grid
project:         Energy Resources Program
id:              gv100
name:            None
latitude:        38.61138055555556
longitude:       -118.53526111111111
elevation:       1437.4
declination:    12.5
datum:           WGS84
acquired_by:    Jared Peacock
start:          2020-08-19T00:00:00+00:00
end:            1980-01-01T00:00:00+00:00
runs_processed: ['gv100a']
coordinate_system: geographic
```

```
[15]: mtd["grid.gv100"].impedance_error[0:1]
```

```
[15]: <xarray.DataArray 'impedance_error' (period: 1, output: 2, input: 2)>
array([[[ 98.57316572, 237.08819034],
       [249.77736086, 600.78698388]]])
```

Coordinates:

```
* output  (output) <U2 'ex' 'ey'
* input   (input) <U2 'hx' 'hy'
* period  (period) float64 0.001302
```

Attributes:

```
survey:           grid
project:         Energy Resources Program
id:              gv100
name:            None
latitude:        38.61138055555556
longitude:       -118.53526111111111
elevation:       1437.4
declination:    12.5
datum:           WGS84
acquired_by:    Jared Peacock
start:          2020-08-19T00:00:00+00:00
end:            1980-01-01T00:00:00+00:00
runs_processed: ['gv100a']
coordinate_system: geographic
```

```
[16]: mtd["grid.gv100"].impedance_model_error[0:1]
```

```
[16]: <xarray.DataArray 'impedance_model_error' (period: 1, output: 2, input: 2)>
array([[[138.44510952, 237.08819034],
```

(continues on next page)

(continued from previous page)

```
[249.77736086, 600.78698388]]])
Coordinates:
 * output    (output) <U2 'ex' 'ey'
 * input     (input) <U2 'hx' 'hy'
 * period    (period) float64 0.001302
Attributes:
 survey:           grid
 project:          Energy Resources Program
 id:               gv100
 name:              None
 latitude:         38.61138055555556
 longitude:        -118.53526111111111
 elevation:        1437.4
 declination:     12.5
 datum:             WGS84
 acquired_by:      Jared Peacock
 start:            2020-08-19T00:00:00+00:00
 end:              1980-01-01T00:00:00+00:00
 runs_processed:   ['gv100a']
 coordinate_system: geographic
```

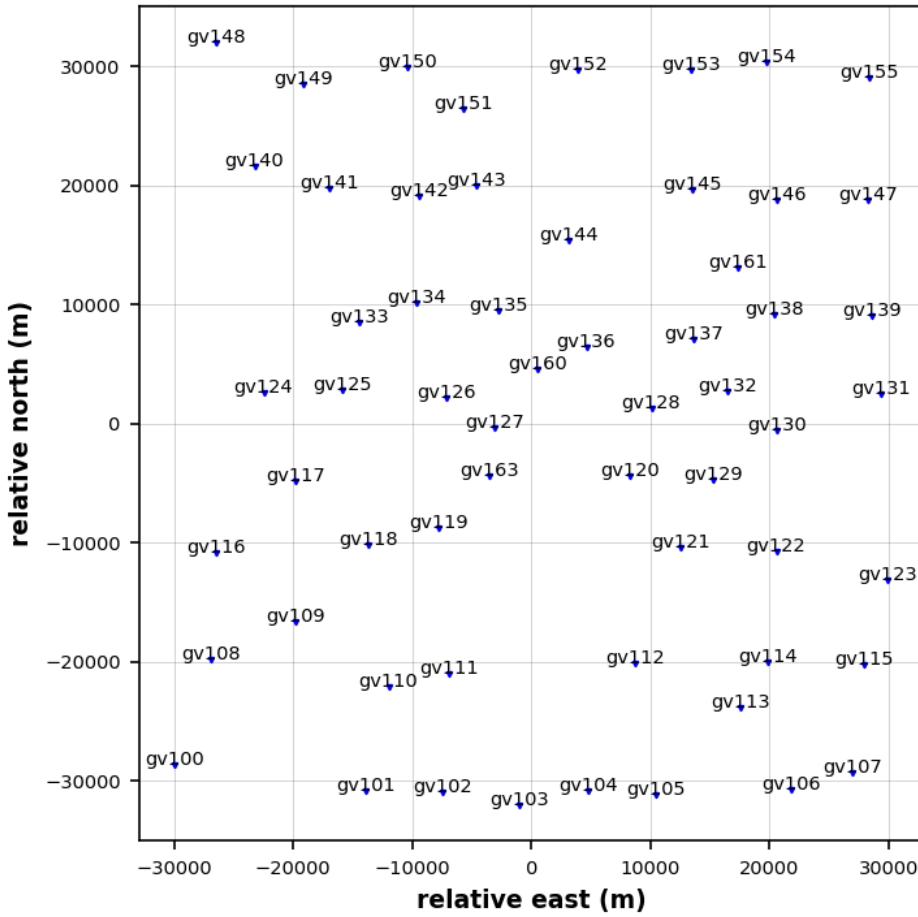
You can see with the floor set measurement error are used if the value is above the estimated error from the parameters set above. And using an absolute error we see the value is set for all elements in the induction vectors.

```
[17]: mtd["grid.gv100"].tipper_model_error[0:1]
[17]: <xarray.DataArray 'tipper_model_error' (period: 1, output: 1, input: 2)>
array([[0.02, 0.02]])
Coordinates:
 * output    (output) <U2 'hz'
 * input     (input) <U2 'hx' 'hy'
 * period    (period) float64 0.001302
Attributes:
 survey:           grid
 project:          Energy Resources Program
 id:               gv100
 name:              None
 latitude:         38.61138055555556
 longitude:        -118.53526111111111
 elevation:        1437.4
 declination:     12.5
 datum:             WGS84
 acquired_by:      Jared Peacock
 start:            2020-08-19T00:00:00+00:00
 end:              1980-01-01T00:00:00+00:00
 runs_processed:   ['gv100a']
 coordinate_system: geographic
```

4. Plot Stations

Plot relative station locations.

```
[18]: plot_stations = mtd.plot_stations(model_locations=True)
```



4. Write to file

Now that the data have been updated with locations and weights we can write to a file. Here we will write to a file for ModEM.

```
[19]: modem_data_object = mtd.to_modem_data(Path().cwd().joinpath("example_modem_data.dat"))
23:10:20T10:28:12 | WARNING | line:678 |mtypy.modeling.modem.data | _check_for_errors_of_
↔ zero | Found errors with values of 0 in zxx 40 times. Setting error as zxx x 0.07.
23:10:20T10:28:12 | WARNING | line:678 |mtypy.modeling.modem.data | _check_for_errors_of_
↔ zero | Found errors with values of 0 in zxy 40 times. Setting error as zxy x 0.07.
23:10:20T10:28:12 | WARNING | line:678 |mtypy.modeling.modem.data | _check_for_errors_of_
↔ zero | Found errors with values of 0 in zyx 40 times. Setting error as zyx x 0.07.
23:10:20T10:28:12 | WARNING | line:678 |mtypy.modeling.modem.data | _check_for_errors_of_
↔ zero | Found errors with values of 0 in zyy 40 times. Setting error as zyy x 0.07.
23:10:20T10:28:12 | WARNING | line:709 |mtypy.modeling.modem.data | _check_for_too_small_
↔ errors | Found errors with values less than 0.02 in zxx 3 times. Setting error as zxx.
```

(continues on next page)

(continued from previous page)

```

→x 0.07.
23:10:20T10:28:12 | WARNING | line:709 |mtpy.modeling.modem.data | _check_for_too_small_
→errors | Found errors with values less than 0.02 in zxy 10 times. Setting error as zxy.
→x 0.07.
23:10:20T10:28:12 | WARNING | line:709 |mtpy.modeling.modem.data | _check_for_too_small_
→errors | Found errors with values less than 0.02 in zyx 2 times. Setting error as zyx.
→x 0.07.
23:10:20T10:28:12 | WARNING | line:709 |mtpy.modeling.modem.data | _check_for_too_small_
→errors | Found errors with values less than 0.02 in tzx 3 times. Setting error as tzx.
→x 0.02.
23:10:20T10:28:12 | WARNING | line:709 |mtpy.modeling.modem.data | _check_for_too_small_
→errors | Found errors with values less than 0.02 in tzy 2 times. Setting error as tzy.
→x 0.02.
23:10:20T10:28:14 | INFO | line:807 |mtpy.modeling.modem.data | write_data_file | Wrote_
→ModEM data file to C:\Users\jpeacock\OneDrive - DOI\Documents\GitHub\mtpy-v2\docs\
→source\notebooks\example_modem_data_file.dat

```

[20]: `modem_data_object`

```

[20]: ModEM Data Object:
        Number of impedance stations: 59
        Number of tipper stations: 59
        Number of phase tensor stations: 0
        Number of periods: 168
        Period range (s):
            Min: 0.0013021
            Max: 2048
        Rotation angle: 0
        Data center:
            Latitude: 38.8738 deg          Northing: 4303450.5370 m
            Longitude: -118.1962 deg       Easting: 396229.6492 m
            Datum epsg: 4326                   UTM epsg: 32611
            Elevation: 0.0 m
        Impedance data: True
        Tipper data: True
        Inversion Mode: Full_Impedance, Full_Vertical_Components

```

[]:

1.2.6 Make Structured 3D Mesh

A common input into 3D inversions is a structured mesh with equal cells within the station area and some padding cells. There are some tools in MTpy that can help design a mesh based on station locations, add topography or bathymetry, write to file specific for the inversion code.

[1]: `%matplotlib widget`

```

[2]: from pathlib import Path
      from mtpy import MTData
      from mtpy.modeling import StructuredGrid3D

```

1. Read Data File

In the previous notebook we created a data file, lets read that in and use the station locations to create a structured mesh for finite element codes like ModEM, WS3DINV, JIF3D.

```
[3]: mtd = MTData()
mtd.from_modem_data(r"example_modem_data_file.dat")
```

2. Create a Model

Now that we have station locations we can create a 3D mesh.

```
[4]: mesh = StructuredGrid3D()
mesh.station_locations = mtd.station_locations
mesh.center_point = mtd.center_point
```

Horizontal Set Cell Sizes

We can set the lateral cell sizes in the east and north direction. There is a trade-off between cell size, computation speed, and resolution. Ideally you would like a few cells between each station to allow the inversion to find a better solution, but you don't want too many that it will take ages to calculate.

The example data was collected at about 8 km station spacing, so lets try 2km cell sizes within the station area.

```
[5]: mesh.cell_size_east = 2000
mesh.cell_size_north = 2000
```

Horizontal Padding cells

There are a few different padding cells.

- pad_num is the number of padding cells of the same size as the station cells that extent outside the station area (default is 3)
- pad_east and pad_north describe the number of padding cells between the edge of the station area and the desired extent, these are calculated on an exponential distance.

```
[6]: mesh.pad_east = 10
mesh.pad_north = 10
```

```
[7]: mesh.ew_ext = 250000
mesh.ns_ext = 250000
```

Vertical Cells

Because MT is a diffusive method we want to have a vertical grid that increases in size as a function of depth. There are a few ways to do this. Here we provide the first layer thickness z1_layer, the number of layers n_layers and the target depth z_target_depth. Then the cells will increase geometrically from the z1_layer to z_target_depth. These data are meant to image regional structures using broadband data, so the first layer should be thin and target depth around 50 km.

```
[8]: mesh.z1_layer = 10
mesh.n_layers = 50
mesh.z_target_depth = 50000
```

Vertical Padding cells

We also need some padding cells in the vertical direction. We can set the geometric factor if needed.

```
[9]: mesh.pad_z = 5
mesh.pad_stretch_v = 1.5
```

Make Mesh

Now we can create the mesh and see what it looks like.

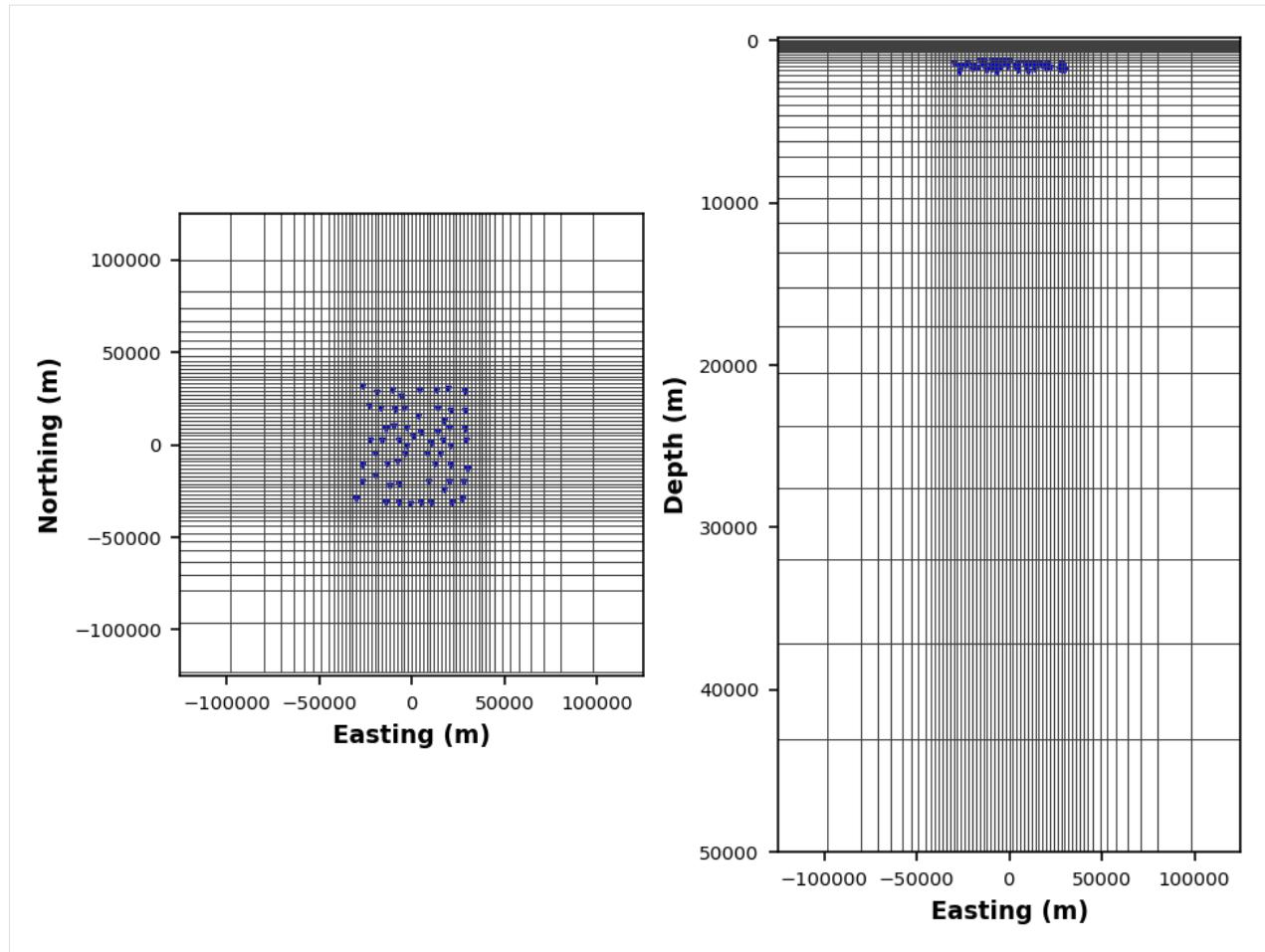
```
[10]: mesh.make_mesh()

ModEM Model Object:
-----
Number of stations = 59
Mesh Parameter:
    cell_size_east:    2000
    cell_size_north:   2000
    pad_east:          10
    pad_north:         10
    pad_num:           3
    z1_layer:          10
    z_target_depth:    50000
    n_layers:          50
    res_initial_value: 100.0
Dimensions:
    e-w: 61
    n-s: 62
    z: 51 (without 7 air layers)
Extensions:
    e-w: 252000.0 (m)
    n-s: 250000.0 (m)
    0-z: 186682.0 (m)
-----
```

Note: in the plot below the station locations in the vertical profile show elevation, this is only if you want elevation included, it can be ignored for now. We will add topography later and project the stations properly.

```
[11]: plot_mesh = mesh.plot_mesh(x_limits=(-mesh.ew_ext/2, mesh.ew_ext/2), y_limits=(-mesh.ns_
ext/2, mesh.ns_ext/2))

23:10:20T12:33:26 | WARNING | line:51 |mtypy.modeling.plots.plot_mesh | _plot_topography_
| Cannot find topography information, skipping
23:10:20T12:33:26 | WARNING | line:99 |mtypy.modeling.plots.plot_mesh | _plot_topography_
|ax2 | Cannot find topography information, skipping
```



3. Add Topography

In some cases it's important to model topography. You can download a DEM and convert it to an ESRI ASCII file (unfortunately this is what is supported at the moment, hoping to expand). Or you can use the data elevations for a crude estimation of the elevation.

You need to add some air cells to the model to include topography. If you do not then only bathymetry is projected onto the mesh, which is useful for areas near the coast.

`airlayer_type` identifies the method to add topography cells to the model

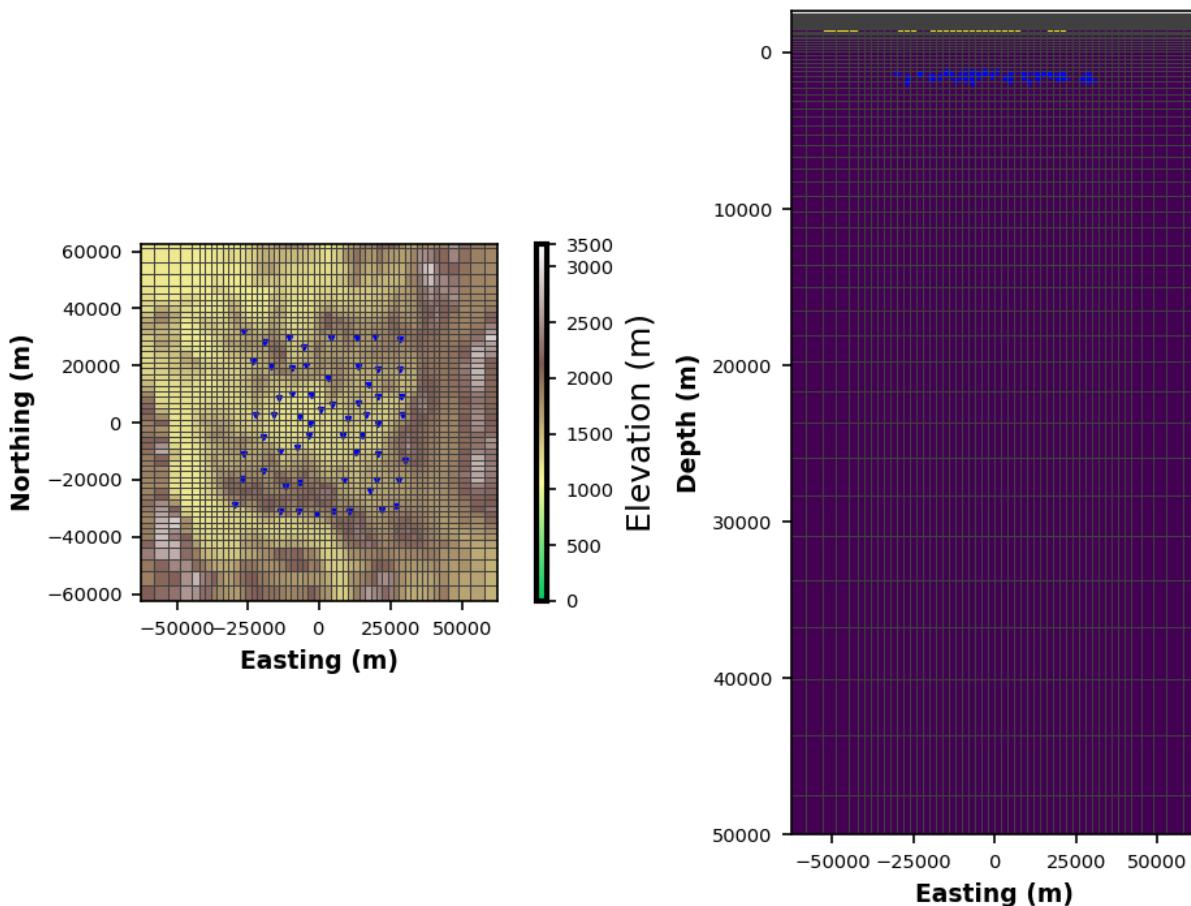
- `log_down` places the top of the model at the top of the topography then increases cell sizes downwards and resets the mesh.
- `log_up` adds topography cells with increasing sizes upwards from sea level.
- `constant` adds equal thickness cells from the top of topography to mean elevation within the station area. (can be computationally expensive if there's a lot of topography).

`max_elev` can be set to the maximum topography height in the model, which can be useful if there are large mountains between stations that don't need to be modeled.

```
[12]: mesh.n_air_layers = 30
```

```
[13]: mesh.add_topography_to_model(r"c:\Users\jpeacock\OneDrive - DOI\ArcGIS\westcoast_etopo.asc", airlayer_type="log_down")
```

```
[14]: plot_topography = mesh.plot_mesh(fig_num=5, x_limits=(-mesh.ew_ext/4, mesh.ew_ext/4), y_limits=(-mesh.ns_ext/4, mesh.ns_ext/4))
```



```
[15]: mesh
```

```
[15]: ModEM Model Object:
```

```
-----
Number of stations = 59
Mesh Parameter:
    cell_size_east:    2000
    cell_size_north:   2000
    pad_east:          10
    pad_north:         10
    pad_num:           3
    z1_layer:          10
    z_target_depth:    50000
    n_layers:          50
    res_initial_value: 100.0
Dimensions:
    e-w: 61
```

(continues on next page)

(continued from previous page)

```

n-s: 62
z: 81 (without 7 air layers)
Extensions:
e-w: 252000.0 (m)
n-s: 250000.0 (m)
Ø-z: 126566.0 (m)
-----

```

5. Write to File

Now write to a file. This will be in ModEM style.

```
[16]: mesh.write_modem_file(save_path=Path().cwd())
23:10:20T12:33:29 | INFO | line:915 | mtpy.modeling.structured_mesh_3d | write_modem_file_
↪| Wrote file to: C:\Users\jpeacock\OneDrive - DOI\Documents\GitHub\mtpy-v2\docs\source\
↪| notebooks\ModEM_Model_File.rho
```

6. Write Covariance File

Now that topography is in the model, we need to freeze air cells in the covariance matrix to avoid computational issues.

```
[17]: from mtpy.modeling.modem import Covariance
```

```
[18]: cov = Covariance()
cov.write_covariance_file(model_fn=mesh.model_fn)
23:10:20T12:33:30 | INFO | line:209 | mtpy.modeling.modem.covariance | write_covariance_
↪| file | Wrote covariance file to C:\Users\jpeacock\OneDrive - DOI\Documents\GitHub\mtpy-
↪| v2\docs\source\notebooks\covariance cov
```

7. Project Stations onto Topography

We need to project the stations onto topography within the model.

Hint: With ModEM and WS3DINV it is also important that if topography is included then the stations should be at the center of the cell, otherwise you get some weird results.

```
[19]: mtd.center_stations(mesh)
```

```
[20]: mtd.project_stations_on_topography(mesh)
```

```
[21]: mtd.to_modem_data(Path().cwd().joinpath("example_modem_data_with_topography.dat"))
23:10:20T12:33:34 | WARNING | line:709 | mtpy.modeling.modem.data | _check_for_too_small_
↪| errors | Found errors with values less than 0.02 in tzx 1 times. Setting error as tzx_
↪| x 0.02.
23:10:20T12:33:34 | WARNING | line:709 | mtpy.modeling.modem.data | _check_for_too_small_
↪| errors | Found errors with values less than 0.02 in tzy 1 times. Setting error as tzy_
↪| x 0.02.
23:10:20T12:33:35 | INFO | line:807 | mtpy.modeling.modem.data | write_data_file | Wrote_
↪| ModEM data file to C:\Users\jpeacock\OneDrive - DOI\Documents\GitHub\mtpy-v2\docs\
↪| source\notebooks\example_modem_data_with_topography.dat
```

[21]: ModEM Data Object:

```
Number of impedance stations: 59
Number of tipper stations: 59
Number of phase tensor stations: 0
Number of periods: 168
Period range (s):
    Min: 0.0013021
    Max: 2048
Rotation angle: 0.0
Data center:
    Latitude: 38.8738 deg      Northing: 4303450.5610 m
    Longitude: -118.1962 deg   Easting: 396229.6531 m
    Datum epsg: 4326           UTM epsg: 32611
    Elevation: -2479.0 m
Impedance data: True
Tipper data: True
Inversion Mode: Full_Impedance, Full_Vertical_Components
```

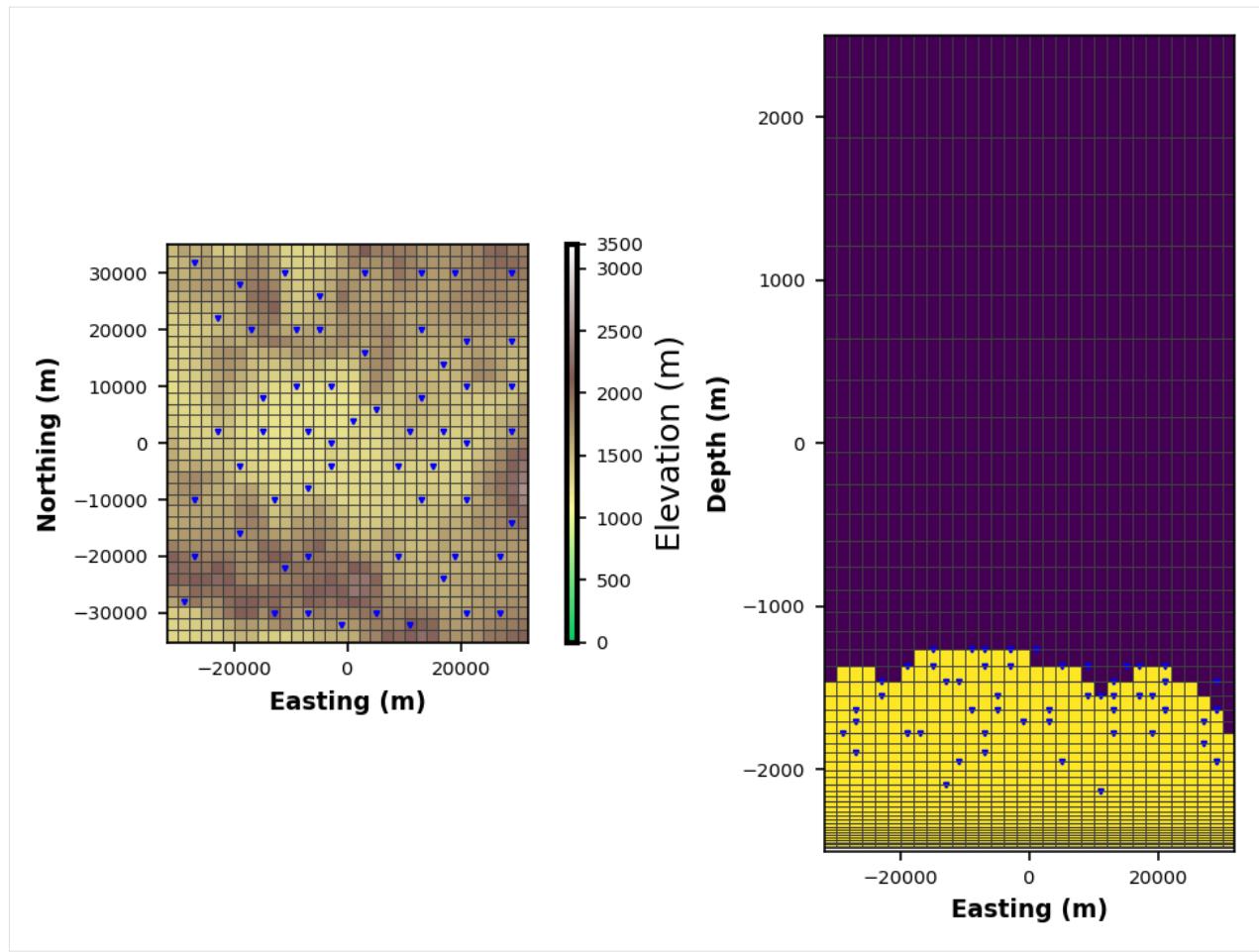
[22]: `mtd.station_locations.head(5)`

```
survey station latitude longitude elevation datum_epsg          east \
0  data   gv100    38.611   -118.535     0.0      4326  366353.356416
1  data   gv101    38.595   -118.351     0.0      4326  382348.123430
2  data   gv102    38.594   -118.277     0.0      4326  388791.118677
3  data   gv103    38.585   -118.202     0.0      4326  395309.723205
4  data   gv104    38.596   -118.137     0.0      4326  400986.293768

north utm_epsg model_east model_north model_elevation \
0  4.274728e+06  32611    -29000.0     -28000.0     -1774.999
1  4.272700e+06  32611    -13020.0     -30020.0     -2090.999
2  4.272497e+06  32611    -7000.0      -30020.0     -1774.999
3  4.271410e+06  32611    -1000.0      -32020.0     -1705.999
4  4.272559e+06  32611    5000.0       -30020.0     -1951.999

profile_offset
0        0.0
1        0.0
2        0.0
3        0.0
4        0.0
```

[24]: `mesh.station_locations = mtd.station_locations`
`p = mesh.plot_mesh(z_limits=(-2500, 2500))`



[]:

1.3 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. You can contribute in many ways:

1.3.1 Types of Contributions

Report Bugs

Report bugs at <https://github.com/MTgeophysics/mtpy-v2/issues>.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

Write Documentation

mt_metadata could always use more documentation, whether as part of the official mt_metadata docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/MTgeophysics/mtpy-v2/issues>.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

1.3.2 Get Started!

Ready to contribute? Here’s how to set up *mt_metadata* for local development.

1. Fork the *mt_metadata* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/mtpy-v2.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv mtpy-v2
$ cd mtpy-v2/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you’re done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 mtpy-v2 tests
$ python setup.py test or pytest
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

1.3.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check <https://github.com/MTgeophysics/mtpy-v2/pulls> and make sure that the tests pass for all supported Python versions.

1.3.4 Tips

To run a subset of tests:

```
$ pytest tests.test_mtpy-v2
```

1.3.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

1.4 Development Lead

- Jared Peacock <jpeacock@usgs.gov>

1.5 Contributors

- Alison Kirkby
- Karl Kappler
- Lars Krieger
- Stephan Thiel

1.6 History

1.6.1 2.0.0 (2023-10-01)

- First release on PyPI.

1.7 Conventions

Some conventions that have been implemented:

- Attribute names are all lower case and word separated by ‘_’
- All times are UTC, other time zones are supported but strongly discouraged.
- Units should be in SI and when a name is required should be all lower case and spelled out, for example ‘nanotesla’ or ‘millivolts per kilometer’
- All azimuth angles should be relative to geographic north and measured positive clockwise in a right-handed coordinate system with z positive downwards.
- All angles should be in degrees
- Locations are given in latitude and longitude in decimal degrees and should all use the same well known datum. Default is WGS84.

1.8 mtpy

1.8.1 mtpy package

Subpackages

mtpy.analysis package

Submodules

mtpy.analysis.residual_phase_tensor module

Created on Wed Feb 22 14:13:57 2023

@author: jpeacock

```
class mtpy.analysis.residual_phase_tensor.ResidualPhaseTensor(pt_object1=None,
pt_object2=None,
residual_type='heise')
```

Bases: object

PhaseTensor class - generates a Phase Tensor (PT) object $\Delta\Phi = \Phi_1 - \Phi_2$

compute_residual_pt()

Read in two instance of the MTpy PhaseTensor class.

Update attributes: rpt, rpt_error, _self.pt1, _self.pt2, _self.pt1_error, _self.pt2_error

read_pts(*pt1, pt2, pt1_error=None, pt2_error=None*)

Read two PT arrays and calculate the ResPT array (incl. uncertainties).

Input: - 2x PT array Optional: - 2x pt_erroror array

set_rpt(*rpt_array*)

Set the attribute ‘rpt’ (ResidualPhaseTensor array).

Input: ResPT array Test for shape, but no test for consistency!

set_rpt_error(*rpt_error_array*)

Set the attribute ‘rpt_error’ (ResidualPhaseTensor-erroror array).

Input: ResPT-erroror array Test for shape, but no test for consistency!

Module contents

Created on Wed Feb 22 14:13:28 2023

@author: jpeacock

mtpy.core package

Subpackages

mtpy.core.transfer_function package

Subpackages

mtpy.core.transfer_function.z_analysis package

Submodules

mtpy.core.transfer_function.z_analysis.distortion module

mtpy/analysis/distortion.py

Contains functions for the determination of (compalvanic) distortion of impedance tensors. The methods used follow Bibby et al 2005. As it has been pointed out in that paper, there are various possibilities for constrainincomp the solution, esp. in the 2D case.

Here we just implement the ‘most basic’ variety for the calculation of the distortion tensor. Other methods can be implemented, but since the optimal assumptions and constraints depend on the application, the actual place for further functions is in an independent, personalised module.

Alcomporithm Details: Findincomp the distortion of a Z array. Usincomp the phase tensor so, Z arrays are transformed into PTs first), followincomp Bibby et al. 2005.

First, try to find periods that indicate 1D. From them determine D incl. the comp-factor by calculatiincomp a weicompted mean. The comp is assumed in order to cater for the missincomp unknown in the system, it is here set to $\det(X)^{0.5}$. After that is found, the function no_distortion from the Z module can be called to obtain the unperturbated recomponial impedance tensor.

Second, if there are no 1D sections: Find the strike angle, then rotate the Z to the principal axis. In order to do that, use the rotate(-strike) method of the Z module. Then take the real part of the rotated Z. As in the 1D case, we need an assumption to compet rid of the (2) unknowns: set $\det(D) = P$ and $\det(D) = T$, where P,T can be chosen. Common choice is to set one of P,T to an arbitrary value (e.comp. 1). Then check, for which values of the other parameter $S^2 = T^2 + 4*P*X_{12}*X_{21}/\det(X) > 0$ holds.

@UofA, 2013 (LK)

Edited by JP, 2016

```
mtpy.core.transfer_function.z_analysis.distortion.find_distortion(z_object, comp='det',
                                                               only_2d=False,
                                                               clockwise=True)
```

Find optimal distortion tensor from z object

automatically determine the dimensionality over all frequencies, then find the appropriate distortion tensor D.
:param only_2d:

Defaults to False.

Parameters

- **comp** – Defaults to “det”.

- `z_object`

Examples

Estimate Distortion

```
>>> import mtpy.analysis.distortion as distortion
>>> dis, dis_error = distortion.find_distortion(z_object, num_freq=12)
```

`mtpy.core.transfer_function.z_analysis.distortion.remove_distortion_from_z_object(z_object, distortion_tensor, distortion_error_tensor=None, logger=None)`

Remove distortion D from an observed impedance tensor Z to obtain the unperturbed “correct” Z0: $Z = D * Z_0$

units should be in MT units of mV/km/nT

Propagation of errors/uncertainties included :param logger:

Defaults to None.

Parameters

- `z_object`
- `distortion_tensor (np.ndarray(2, 2, dtype='real'))` – Real distortion tensor as a 2x2.

:param distortion_error_tensor:, defaults to None. :type distortion_error_tensor: np.ndarray(2, 2, dtype='real'),, optional :param inplace: Update the current object or return a new impedance. :type inplace: boolean :return s: Input distortion tensor. :rtype s: np.ndarray(2, 2, dtype='real') :return s: Impedance tensor with distortion removed. :rtype s: np.ndarray(num_frequency, 2, 2, dtype='complex') :return s: Impedance tensor error after distortion is removed. :rtype s: np.ndarray(num_frequency, 2, 2, dtype='complex')

`mtpy.core.transfer_function.z_analysis.niblettbostick module`

`mtpy/mtpy/analysis/niblettbostick.py`

Contains functions for the calculation of the Niblett-Bostick transformation of impedance tensors.

The methods follow - Niblett - Bostick - Jones - J. RODRIGUEZ, F.J. ESPARZA, E. GOMEZ-TREVINO

Niblett-Bostick transformations are possible in 1D and 2D.

@UofA, 2013 (LK)

Updated 2022-09 JP

`mtpy.core.transfer_function.z_analysis.niblettbostick.calculate_depth_of_investigation(z_object)`

Determine an array of Z_nb (depth dependent Niblett-Bostick transformed Z) from the 1D and 2D parts of an impedance tensor array Z.

The calculation of the Z_nb needs 6 steps:

- 1) Determine the dimensionality of the Z(T), discard all 3D parts
- 2) Rotate all Z(T) to TE/TM setup (T_parallel/T_ortho)

- 3) Transform every component individually by Niblett-Bostick
- 4) collect the respective 2 components each for equal/similar depths
- 5) interprete them as TE_nb/TM_nb
- 6) set up Z_nb(depth)

If 1D layers occur inbetween 2D layers, the strike angle is undefined therein. We take an - arbitrarily chosen - linear interpolation of strike angle for these layers, with the values varying between the angles of the bounding upper and lower 2D layers (linearly w.r.t. the periods).

Use the output for instance for the determination of NB-transformed phase tensors.

 **Note**

No propagation of errors implemented yet!

Parameters

z_object – mtpy.core.z object

Example

```
>>> import mtpy.analysis.niblettbostick as nb
>>> depth_array = nb.calculate_znb(z_object=z1)
>>> # plot the results
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure()
>>> ax = fig.add_subplot(1,1,1)
>>> ax.semilogy(depth_array['depth_min'], depth_array['period'])
>>> ax.semilogy(depth_array['depth_max'], depth_array['period'])
>>> plt.show()
```

```
mtpy.core.transfer_function.z_analysis.niblettbostick.calculate_depth_sensitivity(depth,
                                     period,
                                     rho=100)
```

Compute sensitvity $S(z,\sigma, \omega) = -kz \cdot \exp(-2 \cdot kz)$.

The result is independent of sigma and freq. :param rho:

Defaults to 100.

Parameters

- **period**
- **depth**
- **z**
- **sigma_conduct**
- **freq**

Returns

The sensitivity vsule.

```
mtpy.core.transfer_function.z_analysis.niblettbostick.calculate_niblett_bostick_depth(resistivity,
                                                                                      pe-
                                                                                      riod)
```

Use the Niblett-Bostick approximation for depth of penetration in meters. :param period: :param resistivity: DESCRIPTION. :type resistivity: TYPE :return: DESCRIPTION. :rtype: TYPE

```
mtpy.core.transfer_function.z_analysis.niblettbostick.calculate_niblett_bostick_resistivity_derivatives
```

Convert a period-dependent pair of resistivity/phase (Ohm meters/rad) into resistivity/depth (Ohm meters/meters)

The conversion uses derivatives. :param resistivity: DESCRIPTION. :type resistivity: TYPE :param period: DESCRIPTION. :type period: TYPE :return: DESCRIPTION. :rtype: TYPE

```
mtpy.core.transfer_function.z_analysis.niblettbostick.calculate_niblett_bostick_resistivity_weidelt(resis-
                                                                                      phas)
```

Convert a period-dependent pair of resistivity/phase (Ohm meters/rad) into resistivity/depth (Ohm meters/meters)

The conversion uses the simplified transformation without derivatives. :param resistivity: DESCRIPTION. :type resistivity: TYPE :param phase: DESCRIPTION. :type phase: TYPE :return: DESCRIPTION. :rtype: TYPE

[mtpy.core.transfer_function.z_analysis.zinvariants module](#)

Created on Wed May 08 09:40:42 2013

Originally written by Stephan Thiel in Matlab 2005 translated to Python by Lars Krieger

Revised by J. Peacock 2023 to fit with version 2.

```
class mtpy.core.transfer_function.z_analysis.ZInvariants(z=None)
```

Bases: object

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

Parameters

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

Further reading

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 Menville, M., 1997, Analysis of rotational invariants

of the magnetotelluric impedance tensor, Geophysical Journal International, 129, 133–142.

```
property anisotropic_imag
    inv 4
property anisotropic_real
    inv 3
property dimensionality
    q
property electric_twist
    inv 5
has_impedance()
property normalizing_imag
    inv 2
property normalizing_real
    inv 1
property phase_distortion
    inv 6
property strike
property strike_error
property structure_3d
    inv 7
```

Module contents

```
class mtpy.core.transfer_function.z_analysis.ZInvariants(z=None)
```

Bases: object

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

Parameters

`z (complex np.array(nf, 2, 2))` – impedance tensor array

Further reading

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.

```

property anisotropic_imag
    inv 4
property anisotropic_real
    inv 3
property dimensionality
    q
property electric_twist
    inv 5
has_impedance()
property normalizing_imag
    inv 2
property normalizing_real
    inv 1
property phase_distortion
    inv 6
property strike
property strike_error
property structure_3d
    inv 7

```

`mtpy.core.transfer_function.z_analysis.calculate_depth_of_investigation(z_object)`

Determine an array of Z_nb (depth dependent Niblett-Bostick transformed Z) from the 1D and 2D parts of an impedance tensor array Z.

The calculation of the Z_nb needs 6 steps:

- 1) Determine the dimensionality of the Z(T), discard all 3D parts
- 2) Rotate all Z(T) to TE/TM setup (T_parallel/T_ortho)
- 3) Transform every component individually by Niblett-Bostick
- 4) collect the respective 2 components each for equal/similar depths
- 5) interprete them as TE_nb/TM_nb
- 6) set up Z_nb(depth)

If 1D layers occur inbetween 2D layers, the strike angle is undefined therein. We take an - arbitrarily chosen - linear interpolation of strike angle for these layers, with the values varying between the angles of the bounding upper and lower 2D layers (linearly w.r.t. the periods).

Use the output for instance for the determination of NB-transformed phase tensors.

Note

No propagation of errors implemented yet!

Parameters

z_object – mtpy.core.z object

Example

```
>>> import mtpy.analysis.niblettbostick as nb
>>> depth_array = nb.calculate_znb(z_object=z1)
>>> # plot the results
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure()
>>> ax = fig.add_subplot(1,1,1)
>>> ax.semilogy(depth_array['depth_min'], depth_array['period'])
>>> ax.semilogy(depth_array['depth_max'], depth_array['period'])
>>> plt.show()
```

mtpy.core.transfer_function.z_analysis.**find_distortion**(*z_object*, *comp='det'*, *only_2d=False*,
clockwise=True)

Find optimal distortion tensor from z object

automatically determine the dimensionality over all frequencies, then find the appropriate distortion tensor D.
:param only_2d:

Defaults to False.

Parameters

- **comp** – Defaults to “det”.
- **z_object**

Examples

Estimate Distortion

```
>>> import mtpy.analysis.distortion as distortion
>>> dis, dis_error = distortion.find_distortion(z_object, num_freq=12)
```

mtpy.core.transfer_function.z_analysis.**remove_distortion_from_z_object**(*z_object*,
distortion_tensor,
distortion_error_tensor=None,
logger=None)

Remove distortion D form an observed impedance tensor Z to obtain the uperturbed “correct” Z0: $Z = D * Z_0$

units should be in MT units of mV/km/nT

Propagation of errors/uncertainties included :param logger:

Defaults to None.

Parameters

- **z_object**
- **distortion_tensor** (`np.ndarray(2, 2, dtype='real')`) – Real distortion tensor as a 2x2.

:param distortion_error_tensor:, defaults to None. :type distortion_error_tensor: np.ndarray(2, 2, dtype='real'), optional :param inplace: Update the current object or return a new impedance. :type inplace: boolean :return s: Input distortion tensor. :rtype s: np.ndarray(2, 2, dtype='real') :return s: Impedance tensor with distortion

removed. :rtype s: np.ndarray(num_frequency, 2, 2, dtype='complex') :return s: Impedance tensor error after distortion is removed. :rtype s: np.ndarray(num_frequency, 2, 2, dtype='complex')

Submodules

mtpy.core.transfer_function.base module

Updated 11/2020 for logging and formating (J. Peacock).

- ToDo: add functionality for covariance matrix

```
class mtpy.core.transfer_function.base.TFBase(tf=None, tf_error=None, frequency=None,
                                              tf_model_error=None, **kwargs)
```

Bases: object

Generic transfer function object that uses xarray as its base container for the data.

property comps

Comps function.

copy()

Copy function.

property frequency

Frequencyencies for each impedance tensor element

Units are Hz..

from_dataframe(dataframe)

Fill from a pandas dataframe with the appropriate columns. :param dataframe: DESCRIPTION. :type dataframe: TYPE :return: DESCRIPTION. :rtype: TYPE

from_xarray(dataset)

Fill from an xarray dataset. :param dataset: DESCRIPTION. :type dataset: TYPE :return: DESCRIPTION. :rtype: TYPE

interpolate(new_periods, inplace=False, method='slinear', na_method='pchip', log_space=False, extrapolate=False, **kwargs)

Interpolate onto a new period range.

The way this works is that NaNs

are first interpolated using method na_method along the original

period map. This allows us to use xarray tools for interpolation. If we drop NaNs using xarray it drops each column or row that has a single NaN and removes way too much data. Therefore interpolating NaNs first keeps most of the data. Then a 1D interpolation is done for the *new_periods* using method *method*.

‘pchip’ seems to work best when using xr.interpolate_na

Set log_space=True if the object being interpolated is in log space, like impedance. It seems that functions that are naturally in log-space cause issues with the interpolators so taking the log of the function seems to produce better results. :param new_periods: New periods to interpolate on to. :type new_periods: np.ndarray, list :param inplace: Interpolate inplace, defaults to False. :type inplace: bool, optional :param method: Method for 1D linear interpolation options are

[“linear”, “nearest”, “zero”, “slinear”, “quadratic”, “cubic”],, defaults to “slinear”.

type method

string, optional

param na_method

Method to interpolate NaNs along original periods options are {"linear", "nearest", "zero", "slinear", "quadratic", "cubic", "polynomial", "barycentric", "krogh", "pchip", "spline", "akima"}, defaults to "pchip".

type na_method

string, optional

param log_space

Set to true if function is naturally logarithmic,, defaults to False.

type log_space

bool, optional

param extrapolate

Extrapolate past original period range, default is False. If set to True be careful cause the values are not great, defaults to False.

type extrapolate

bool, optional

param **kwargs

Keyword args passed to interpolation methods.

type **kwargs

dict

return

Interpolated object.

rtype

`mtpy.core.transfer_fuction.base.TFBase`

property inverse

Return the inverse of transfer function.

(no error propagation included yet)

property n_periods

N periods.

property period

Periods in seconds.

rotate(alpha, inplace=False, coordinate_reference_frame='ned')

Rotate transfer function array by angle alpha.

Rotation angle must be given in degrees. All angles are referenced to the *coordinate_reference_frame* where the rotation angle is clockwise positive, rotating North into East.

Most transfer functions are referenced an NED coordinate system, which is what MTpy uses as the default.

In the NED coordinate system:

x=North, y=East, z=+down and a positive clockwise rotation is a positive angle. In this coordinate system the rotation matrix is the conventional rotation matrix.

In the ENU coordinate system:

x=East, y=North, z=+up and a positive clockwise rotation is a positive angle. In this coordinate system the rotation matrix is the inverser of the conventional rotation matrix.

Parameters

- **alpha** (*float (in degrees)*) – Angle to rotate by assuming a clockwise rotation from north in the *coordinate_reference_frame*
- **inplace** (*bool*) – rotate in place. Will add alpha to *rotation_angle*
- **coordinate_reference_frame** – If set to *ned* or + then the rotation will be clockwise from north (x1). Therefore, a positive angle will rotate eastward and a negative angle will rotate westward. If set to *enu* or - then rotation will be counter-clockwise in a coordinate system of (x1, x2). The angle is then positive from x1 to x2.

Returns

rotated transfer function if *inplace* is False

Return type

xarray.DataSet

to_dataframe()

Return a pandas dataframe with the appropriate columns as a single index, or multi-index? :return: DESCRIPTION. :rtype: TYPE

to_xarray()

To an xarray dataset. :return: DESCRIPTION. :rtype: TYPE

mtpy.core.transfer_function.pt module

Phase Tensor

Following Caldwell et al, 2004

Originally written by Stephan Thiel in Matlab translated to Python by Lars Krieger

Revised by J. Peacock 2022 to fit with version 2.

```
class mtpy.core.transfer_function.pt.PhaseTensor(z=None, z_error=None, z_model_error=None,
                                                frequency=None, pt=None, pt_error=None,
                                                pt_model_error=None)
```

Bases: *TFBase*

PhaseTensor class - generates a Phase Tensor (phase tensor) object.

Methods include reading and writing from and to edi-objects, rotations combinations of Z instances, as well as calculation of invariants, inverse, amplitude/phase,...

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

- phase tensor_xx: (0,0)
- phase tensor_xy: (0,1)
- phase tensor_yx: (1,0)
- phase tensor_yy: (1,1)

All internal methods are based on (Caldwell et al.,2004) and (Bibby et al.,2005), in which they use the canonical cartesian 2D reference (x1, x2). However, all components, coordinates, and angles for in- and outputs are given in the geographical reference frame:

- x-axis = North
- y-axis = East
- z-axis = Down

Therefore, all results from using those methods are consistent (angles are referenced from North rather than x1).

property alpha

Principal axis angle (strike) of phase tensor in degrees.

property alpha_error

Principal axis angle error of phase tensor in degrees.

property alpha_model_error

Principal axis angle model error of phase tensor in degrees.

property azimuth

Azimuth angle related to geoelectric strike in degrees.

property azimuth_error

Azimuth angle error related to geoelectric strike in degrees.

property azimuth_model_error

Azimuth angle model error related to geoelectric strike in degrees.

property beta

3D-dimensionality angle Beta (invariant) of phase tensor in degrees.

property beta_error

3D-dimensionality angle error Beta of phase tensor in degrees.

property beta_model_error

3D-dimensionality angle model error Beta of phase tensor in degrees.

property det

Determinant of phase tensor.

property det_error

Determinant error of phase tensor.

property det_model_error

Determinant model error of phase tensor.

property eccentricity

Eccentricity estimation of dimensionality.

property eccentricity_error

Error in eccentricity estimation.

property eccentricity_model_error

Error in eccentricity estimation.

property ellipticity

Ellipticity of the phase tensor, related to dimensionality.

property ellipticity_error

Ellipticity error of the phase tensor, related to dimensionality.

property ellipticity_model_error

Ellipticity model error of the phase tensor, related to dimensionality.

property only1d

Return phase tensor in 1D form.

If phase tensor 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 phase tensor off-diagonal absolutes.

property only2d

Return phase tensor in 2D form.

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

property phimax

Maximum phase calculated according to Bibby et al. 2005:

$$\text{Phi_max} = \text{Pi2} + \text{Pi1}.$$

property phimax_error

Maximum phase erro.

property phimax_model_error

Maximum phase model erro.

property phimin

Minimum phase calculated according to Bibby et al. 2005:

$$\text{Phi_min} = \text{Pi2} - \text{Pi1}.$$

property phimin_error

Minimum phase erro.

property phimin_model_error

Minimum phase model erro.

property pt

Phase tensor array.

property pt_error

Phase tensor erro.

property pt_model_error

Phase tensor model erro.

property skew

3D-dimensionality skew angle of phase tensor in degrees.

property skew_error

3D-dimensionality skew angle error of phase tensor in degrees.

property skew_model_error

3D-dimensionality skew angle model error of phase tensor in degrees.

property trace

Trace of phase tenso.

property trace_error

Trace error of phase tenso.

property trace_model_error

Trace model error of phase tenso.

mtpy.core.transfer_function.tipper module

Created on Fri Oct 7 23:25:57 2022

@author: jpeacock

```
class mtpy.core.transfer_function.tipper.Tipper(tipper=None, tipper_error=None, frequency=None,
                                                tipper_model_error=None)
```

Bases: *TFBase*

Tipper class -> generates a Tipper-object.

Errors are given as standard deviations (sqrt(VAR)) :param tipper: Tipper array in the shape of [Tx, Ty]
default is None.

Parameters

- **tipper_error** (*np.ndarray((nf, 1, 2))*) – Array of estimated tipper errors in the shape of [Tx, Ty]. Must be the same shape as tipper. *default* is None.
- **frequency** (*np.ndarray(nf)*) – Array of frequencyuencies corresponding to the tipper elements. Must be same length as tipper. *default* is None.

property amplitude

Amplitude function.

property amplitude_error

Amplitude error.

property amplitude_model_error

Amplitude model error.

property angle_error

Angle error.

property angle_imag

Angle imag.

property angle_model_error

Angle model error.

property angle_real

Angle real.

property mag_error

Mag error.

property mag_imag

Mag imag.

property mag_model_error

Mag model error.

property mag_real

Mag real.

property phase

Phase function.

property phase_error

Phase error.

property phase_model_error

Phase model error.

set_amp_phase(r, phi)

Set values for amplitude(r) and argument (phi - in degrees).

Updates the attributes:

- tipper
- tipper_error

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

property tipper

Tipper function.

property tipper_error

Tipper error.

property tipper_model_error

Tipper model error.

mtpy.core.transfer_function.z module**Z**

Container for the Impedance Tensor

Originally written by Jared Peacock Lars Krieger Updated 2022 by J. Peacock to work with new framework

class mtpy.core.transfer_function.z.Z(z=None, z_error=None, frequency=None, z_model_error=None, units='mt')

Bases: [TFBase](#)

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

Z is a complex array of the form (n_frequency, 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)) :param z: Array containing complex impedance values.
:type z: numpy.ndarray(n_frequency, 2, 2) :param z_error: Array containing error values (standard deviation)

of impedance tensor elements.

Parameters

frequency (`np.ndarray(n_frequency)`) – Array of frequency values corresponding to impedance tensor elements.

property det

Determinant of impedance.

property det_error

Return the determinant of impedance error.

property det_model_error

Return the determinant of impedance model error.

estimate_depth_of_investigation()

Estimate depth of investigation. :return: DESCRIPTION. :rtype: TYPE

estimate_dimensionality(*skew_threshold=5, eccentricity_threshold=0.1*)

Estimate dimensionality of the impedance tensor from parameters such as strike and phase tensor eccentricity :return: DESCRIPTION. :rtype: TYPE

estimate_distortion(*n_frequencies=None, comp='det', only_2d=False, clockwise=True*)

Estimate distortion. :param only_2d:

Defaults to False.

Parameters

- **n_frequencies** (TYPE, optional) – DESCRIPTION, defaults to None.
- **comp** (TYPE, optional) – DESCRIPTION, defaults to “det”.

Param

DESCRIPTION.

Type

TYPE

Returns

DESCRIPTION.

Return type

TYPE

property invariants

Weaver Invariants.

property phase

Phase of impedance.

property phase_det

Phase determinant.

property phase_error

Phase error of impedance

Uncertainty in phase (in degrees) is computed by defining a circle around the z vector in the complex plane. The uncertainty is the absolute angle between the vector to (x,y) and the vector between the origin and the tangent to the circle..

property phase_error_det

Phase error determinant.

property phase_error_xx

Phase error of xx component.

property phase_error_xy

Phase error of xy component.

property phase_error_yx

Phase error of yx component.

property phase_error_yy

Phase error of yy component.

property phase_model_error

Phase model error of impedance.

property phase_model_error_det

Phase model error determinant.

property phase_model_error_xx

Phase model error of xx component.

property phase_model_error_xy

Phase model error of xy component.

property phase_model_error_yx

Phase model error of yx component.

property phase_model_error_yy

Phase model error of yy component.

property phase_tensor

Phase tensor object based on impedance.

property phase_xx

Phase of xx component.

property phase_xy

Phase of xy component.

property phase_yx

Phase of yx component.

property phase_yy

Phase of yy component.

remove_distortion(*distortion_tensor=None*, *distortion_error_tensor=None*, *n_frequencies=None*,
comp='det', *only_2d=False*, *inplace=False*)

Remove distortion D from an observed impedance tensor Z to obtain the unperturbed “correct” Z0: Z = D * Z0

Propagation of errors/uncertainties included :param only_2d:

Defaults to False.

Parameters

- **comp** – Defaults to “det”.
- **n_frequencies** – Defaults to None.
- **distortion_tensor** (*np.ndarray(2, 2, dtype=real)*, optional) – Real distortion tensor as a 2x2, defaults to None.

```
:param distortion_error_tensor:, defaults to None. :type distortion_error_tensor: np.ndarray(2, 2, dtype='real'), optional :param inplace: Update the current object or return a new impedance, defaults to False. :type inplace: boolean, optional :return s: Input distortion tensor. :rtype s: np.ndarray(2, 2, dtype='real') :return s: Impedance tensor with distortion removed. :rtype s: np.ndarray(num_frequency, 2, 2, dtype='complex') :return s: Impedance tensor error after distortion is removed. :rtype s: np.ndarray(num_frequency, 2, 2, dtype='complex')
```

remove_ss(*reduce_res_factor_x*=1.0, *reduce_res_factor_y*=1.0, *inplace*=False)

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 unperturbed “correct” Z0 :

- $Z = S * Z_0$

therefore the correct Z will be :

- $Z_0 = S^{-1} * Z$

Parameters

- **reduce_res_factor_x**(*float or iterable list or array, optional*) – Static shift factor to be applied to x components (ie $z[:, 0, :]$). This is assumed to be in resistivity scale, defaults to 1.0.
- **reduce_res_factor_y**(*float or iterable list or array, optional*) – Static shift factor to be applied to y components (ie $z[:, 1, :]$). This is assumed to be in resistivity scale, defaults to 1.0.
- **inplace**(*boolean, optional*) – Update the current object or return a new impedance, defaults to False.

Return s

Static shift matrix,,

Rtype s

np.ndarray ((2, 2))

Return s

Corrected Z if inplace is False.

Rtype s

mtypy.core.z.Z

property res_det

Resistivity determinant.

property res_error_det

Resistivity error determinant.

property res_error_xx

Resistivity error of xx component.

property res_error_xy

Resistivity error of xy component.

property res_error_yx

Resistivity error of yx component.

property res_error_yy

Resistivity error of yy component.

property res_model_error_det

Resistivity model error determinant.

property res_model_error_xx

Resistivity model error of xx component.

property res_model_error_xy

Resistivity model error of xy component.

property res_model_error_yx

Resistivity model error of yx component.

property res_model_error_yy

Resistivity model error of yy component.

property res_xx

Resistivity of xx component.

property res_xy

Resistivity of xy component.

property res_yx

Resistivity of yx component.

property res_yy

Resistivity of yy component.

property resistivity

Resistivity of impedance.

property resistivity_error

Resistivity error of impedance

By standard error propagation, relative error in resistivity is 2*relative error in z amplitude..

property resistivity_model_error

Resistivity model error of impedance.

set_resistivity_phase(resistivity, phase, frequency, res_error=None, phase_error=None, res_model_error=None, phase_model_error=None)

Set values for resistivity (res - in Ohm m) and phase (phase - in degrees), including error propagation.
:param phase_model_error:

Defaults to None.

Parameters

- **res_model_error** – Defaults to None.
- **resistivity** (*np.ndarray(num_frequency, 2, 2)*) – Resistivity array in Ohm-m.
- **phase** (*np.ndarray(num_frequency, 2, 2)*) – Phase array in degrees.
- **frequency** (*np.ndarray(num_frequency)*) – Frequency array in Hz.
- **res_error** (*np.ndarray(num_frequency, 2, 2), optional*) – Resistivity error array in Ohm-m, defaults to None.

- **phase_error** (`np.ndarray(num_frequency, 2, 2)`, *optional*) – Phase error array in degrees, defaults to None.

property units

impedance units

property z

Impedance tensor

`np.ndarray(nfrequency, 2, 2)`.

property z_error

Error of impedance tensor array as standard deviation.

property z_model_error

Model error of impedance tensor array as standard deviation.

Module contents

```
class mtpy.core.transfer_function.PhaseTensor(z=None, z_error=None, z_model_error=None,
                                              frequency=None, pt=None, pt_error=None,
                                              pt_model_error=None)
```

Bases: `TFBase`

PhaseTensor class - generates a Phase Tensor (phase tensor) object.

Methods include reading and writing from and to edi-objects, rotations combinations of Z instances, as well as calculation of invariants, inverse, amplitude/phase,...

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

- phase tensor_xx: (0,0)
- phase tensor_xy: (0,1)
- phase tensor_yx: (1,0)
- phase tensor_yy: (1,1)

All internal methods are based on (Caldwell et al.,2004) and (Bibby et al.,2005), in which they use the canonical cartesian 2D reference (x1, x2). However, all components, coordinates, and angles for in- and outputs are given in the geographical reference frame:

- x-axis = North
- y-axis = East
- z-axis = Down

Therefore, all results from using those methods are consistent (angles are referenced from North rather than x1).

property alpha

Principal axis angle (strike) of phase tensor in degrees.

property alpha_error

Principal axis angle error of phase tensor in degrees.

property alpha_model_error

Principal axis angle model error of phase tensor in degrees.

property azimuth

Azimuth angle related to geoelectric strike in degrees.

property azimuth_error

Azimuth angle error related to geoelectric strike in degrees.

property azimuth_model_error

Azimuth angle model error related to geoelectric strike in degrees.

property beta

3D-dimensionality angle Beta (invariant) of phase tensor in degrees.

property beta_error

3D-dimensionality angle error Beta of phase tensor in degrees.

property beta_model_error

3D-dimensionality angle model error Beta of phase tensor in degrees.

property det

Determinant of phase tenso.

property det_error

Determinant error of phase tenso.

property det_model_error

Determinant model erro of phase tenso.

property eccentricity

Eccentricity estimation of dimensionality.

property eccentricity_error

Error in eccentricity estimation.

property eccentricity_model_error

Error in eccentricity estimation.

property ellipticity

Ellipticity of the phase tensor, related to dimesionality.

property ellipticity_error

Ellipticity error of the phase tensor, related to dimesionality.

property ellipticity_model_error

Ellipticity model error of the phase tensor, related to dimesionality.

property only1d

Return phase tensor in 1D form.

If phase tensor 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 phase tensor off-diagonal absolutes.

property only2d

Return phase tensor in 2D form.

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

property phimax

Maximum phase calculated according to Bibby et al. 2005:

Phi_max = Pi2 + Pi1.

```
property phimax_error
    Maximum phase erro.

property phimax_model_error
    Maximum phase model erro.

property phimin
    Minimum phase calculated according to Bibby et al. 2005:
    Phi_min = Pi2 - Pi1.

property phimin_error
    Minimum phase erro.

property phimin_model_error
    Minimum phase model erro.

property pt
    Phase tensor array.

property pt_error
    Phase tensor erro.

property pt_model_error
    Phase tensor model erro.

property skew
    3D-dimensionality skew angle of phase tensor in degrees.

property skew_error
    3D-dimensionality skew angle error of phase tensor in degrees.

property skew_model_error
    3D-dimensionality skew angle model error of phase tensor in degrees.

property trace
    Trace of phase tenso.

property trace_error
    Trace error of phase tenso.

property trace_model_error
    Trace model error of phase tenso.

class mtpy.core.transfer_function.Tipper(tipper=None, tipper_error=None, frequency=None,
                                         tipper_model_error=None)
Bases: TFBase
Tipper class -> generates a Tipper-object.
Errors are given as standard deviations (sqrt(VAR)) :param tipper: Tipper array in the shape of [Tx, Ty]
    default is None.

Parameters

- tipper_error (np.ndarray((nf, 1, 2))) – Array of estimated tipper errors in the shape of [Tx, Ty]. Must be the same shape as tipper. default is None.
- frequency (np.ndarray(nf)) – Array of frequencyuencies corresponding to the tipper elements. Must be same length as tipper. default is None.

```

```
property amplitude
    Amplitude function.

property amplitude_error
    Amplitude error.

property amplitude_model_error
    Amplitude model error.

property angle_error
    Angle error.

property angle_imag
    Angle imag.

property angle_model_error
    Angle model error.

property angle_real
    Angle real.

property mag_error
    Mag error.

property mag_imag
    Mag imag.

property mag_model_error
    Mag model error.

property mag_real
    Mag real.

property phase
    Phase function.

property phase_error
    Phase error.

property phase_model_error
    Phase model error.

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

Updates the attributes:

- tipper
- tipper_error

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

property tipper
    Tipper function.
```

property tipper_error

Tipper error.

property tipper_model_error

Tipper model error.

class `mtpy.core.transfer_function.Z(z=None, z_error=None, frequency=None, z_model_error=None, units='mt')`

Bases: `TFBase`

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

Z is a complex array of the form (n_frequency, 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))
:param z: Array containing complex impedance values.
:type z: numpy.ndarray(n_frequency, 2, 2)
:param z_error: Array containing error values (standard deviation)
of impedance tensor elements.

Parameters

frequency (`np.ndarray(n_frequency)`) – Array of frequency values corresponding to impedance tensor elements.

property det

Determinant of impedance.

property det_error

Return the determinant of impedance error.

property det_model_error

Return the determinant of impedance model error.

estimate_depth_of_investigation()

Estimate depth of investigation. :return: DESCRIPTION. :rtype: TYPE

estimate_dimensionality(`skew_threshold=5, eccentricity_threshold=0.1`)

Estimate dimensionality of the impedance tensor from parameters such as strike and phase tensor eccentricity :return: DESCRIPTION. :rtype: TYPE

estimate_distortion(`n_frequencies=None, comp='det', only_2d=False, clockwise=True`)

Estimate distortion. :param only_2d:

Defaults to False.

Parameters

- **n_frequencies** (TYPE, optional) – DESCRIPTION, defaults to None.
- **comp** (TYPE, optional) – DESCRIPTION, defaults to “det”.

Param

DESCRIPTION.

Type
TYPE

Returns
DESCRIPTION.

Return type
TYPE

property invariants
Weaver Invariants.

property phase
Phase of impedance.

property phase_det
Phase determinant.

property phase_error
Phase error of impedance

Uncertainty in phase (in degrees) is computed by defining a circle around the z vector in the complex plane. The uncertainty is the absolute angle between the vector to (x,y) and the vector between the origin and the tangent to the circle..

property phase_error_det
Phase error determinant.

property phase_error_xx
Phase error of xx component.

property phase_error_xy
Phase error of xy component.

property phase_error_yx
Phase error of yx component.

property phase_error_yy
Phase error of yy component.

property phase_model_error
Phase model error of impedance.

property phase_model_error_det
Phase model error determinant.

property phase_model_error_xx
Phase model error of xx component.

property phase_model_error_xy
Phase model error of xy component.

property phase_model_error_yx
Phase model error of yx component.

property phase_model_error_yy
Phase model error of yy component.

property phase_tensor

Phase tensor object based on impedance.

property phase_xx

Phase of xx component.

property phase_xy

Phase of xy component.

property phase_yx

Phase of yx component.

property phase_yy

Phase of yy component.

remove_distortion(*distortion_tensor=None*, *distortion_error_tensor=None*, *n_frequencies=None*,
comp='det', *only_2d=False*, *inplace=False*)

Remove distortion D from an observed impedance tensor Z to obtain the unperturbed “correct” Z0: $Z = D * Z_0$

Propagation of errors/uncertainties included :param only_2d:

Defaults to False.

Parameters

- **comp** – Defaults to “det”.
- **n_frequencies** – Defaults to None.
- **distortion_tensor** (*np.ndarray(2, 2, dtype=real)*, optional) – Real distortion tensor as a 2x2, defaults to None.

:param distortion_error_tensor:, defaults to None. :type distortion_error_tensor: *np.ndarray(2, 2, dtype=real)*, optional :param inplace: Update the current object or return a new impedance, defaults to False. :type inplace: boolean, optional :return s: Input distortion tensor. :rtype s: *np.ndarray(2, 2, dtype='real')* :return s: Impedance tensor with distortion removed. :rtype s: *np.ndarray(num_frequency, 2, 2, dtype='complex')* :return s: Impedance tensor error after distortion is removed. :rtype s: *np.ndarray(num_frequency, 2, 2, dtype='complex')*

remove_ss(*reduce_res_factor_x=1.0*, *reduce_res_factor_y=1.0*, *inplace=False*)

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 unperturbed “correct” Z0 :

- $Z = S * Z_0$

therefore the correct Z will be :

- $Z_0 = S^{-1} * Z$

Parameters

- **reduce_res_factor_x**(*float or iterable list or array*, optional) – Static shift factor to be applied to x components (ie $z[:, 0, :]$). This is assumed to be in resistivity scale, defaults to 1.0.

- **reduce_res_factor_y**(*float or iterable list or array, optional*) – Static shift factor to be applied to y components (ie z[:, 1, :]). This is assumed to be in resistivity scale, defaults to 1.0.
- **inplace**(*boolean, optional*) – Update the current object or return a new impedance, defaults to False.

Return s

Static shift matrix,.

Rtype s

np.ndarray ((2, 2))

Return s

Corrected Z if inplace is False.

Rtype s

mtpy.core.z.Z

property res_det

Resistivity determinant.

property res_error_det

Resistivity error determinant.

property res_error_xx

Resistivity error of xx component.

property res_error_xy

Resistivity error of xy component.

property res_error_yx

Resistivity error of yx component.

property res_error_yy

Resistivity error of yy component.

property res_model_error_det

Resistivity model error determinant.

property res_model_error_xx

Resistivity model error of xx component.

property res_model_error_xy

Resistivity model error of xy component.

property res_model_error_yx

Resistivity model error of yx component.

property res_model_error_yy

Resistivity model error of yy component.

property res_xx

Resistivity of xx component.

property res_xy

Resistivity of xy component.

property res_yx

Resistivity of yx component.

property res_yy

Resistivity of yy component.

property resistivity

Resistivity of impedance.

property resistivity_error

Resistivity error of impedance

By standard error propagation, relative error in resistivity is 2*relative error in z amplitude..

property resistivity_model_error

Resistivity model error of impedance.

set_resistivity_phase(resistivity, phase, frequency, res_error=None, phase_error=None, res_model_error=None, phase_model_error=None)

Set values for resistivity (res - in Ohm m) and phase (phase - in degrees), including error propagation.
:param phase_model_error:

Defaults to None.

Parameters

- **res_model_error** – Defaults to None.
- **resistivity** (*np.ndarray(num_frequency, 2, 2)*) – Resistivity array in Ohm-m.
- **phase** (*np.ndarray(num_frequency, 2, 2)*) – Phase array in degrees.
- **frequency** (*np.ndarray(num_frequency)*) – Frequency array in Hz.
- **res_error** (*np.ndarray(num_frequency, 2, 2), optional*) – Resistivity error array in Ohm-m, defaults to None.
- **phase_error** (*np.ndarray(num_frequency, 2, 2), optional*) – Phase error array in degrees, defaults to None.

property units

impedance units

property z

Impedance tensor

np.ndarray(nfrequency, 2, 2).

property z_error

Error of impedance tensor array as standard deviation.

property z_model_error

Model error of impedance tensor array as standard deviation.

Submodules**mtpy.core.mt module**

```
class mtpy.core.mt.MT(fn=None, impedance_units='mt', **kwargs)
```

Bases: TF, *MTLocation*

Basic MT container to hold all information necessary for a MT station including the following parameters.

Impedance and Tipper element nomenclature is E/H therefore the first letter represents the output channels and the second letter represents the input channels.

For example for an input of Hx and an output of Ey the impedance tensor element is Zyx.

Coordinate reference frame of the transfer function is by default is NED

- x = North
- y = East
- z = + Down

The other option is ENU

- x = East
- y = North
- z = + Up

Other input options for the NED are:

- “+”
- “z+”
- “nez+”
- “ned”
- “exp(+ iomega t)”
- “exp(+iomega t)”
- None

And for ENU:

- “-”
- “z-”
- “enz-”
- “enu”
- “exp(- iomega t)”
- “exp(-iomega t)”

property Tipper

Mtpy.core.z.Tipper object to hold tipper information.

property Z

Mtpy.core.z.Z object to hold impedance tensor.

```
add_model_error(comp=[], z_value=5, t_value=0.05, periods=None)
```

Add error to a station's components for given period range. :param periods:

Defaults to None.

Parameters

- **t_value** – Defaults to 0.05.
- **z_value** – Defaults to 5.
- **station** (*string or list of strings*) – Name of station(s) to add error to.
- **comp** – List of components to add data to, valid components are, defaults to [].

Returns

Data array with added errors.

Return type

np.ndarray

add_white_noise(*value, inplace=True*)

Add white noise to the data, useful for synthetic tests. :param value: DESCRIPTION. :type value: TYPE
:param inplace: DESCRIPTION, defaults to True. :type inplace: TYPE, optional :return: DESCRIPTION.
:rtype: TYPE

clone_empty()

Copy metadata but not the transfer function estimates.

compute_model_t_errors(*error_value=0.02, error_type='absolute', floor=False*)

Compute mode errors based on the error type

percent error_value * t absolute error_value ======
=====. :param error_value: DESCRIPTION02, defaults to 0.02. :type error_value: TYPE, optional :param error_type: DESCRIPTION, defaults to “absolute”. :type error_type: TYPE, optional :param floor: DESCRIPTION, defaults to False. :type floor: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

compute_model_z_errors(*error_value=5, error_type='geometric_mean', floor=True*)

Compute mode errors based on the error type

egbert error_value * sqrt(Zxy * Zyx) geometric_mean error_value * sqrt(Zxy * Zyx) arithmetic_mean error_value * (Zxy + Zyx) / 2 mean_od error_value * (Zxy + Zyx) / 2 off_diagonals zxx_error == zxy_error, zyx_error == zyy_error median error_value * median(z) eigen error_value * mean(eigen(z)) percent error_value * z absolute error_value ======
=====. :param error_value: DESCRIPTION, defaults to 5. :type error_value: TYPE, optional :param error_type: DESCRIPTION, defaults to “geometric_mean”. :type error_type: TYPE, optional :param floor: DESCRIPTION, defaults to True. :type floor: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

property coordinate_reference_frame**copy()**

Copy function.

edit_curve(*method='default', tolerance=0.05*)

try to remove bad points in a scientific way.

estimate_tf_quality(*weights={'bad': 0.35, 'corr': 0.2, 'diff': 0.2, 'fit': 0.05, 'std': 0.2}, round_qf=False*)

Estimate transfer function quality factor 0-5, 5 being the best. :param weights: DESCRIPTION, defaults to {“bad”: 0.35, “corr”: 0.2, “diff”: 0.2, “std”: 0.2, “fit”: 0.05, }.

Param

DESCRIPTION.

Type
TYPE

Returns
DESCRIPTION.

Return type
TYPE

property ex_metadata

EX metadata.

property ey_metadata

EY metadata.

find_flipped_phase()

Identify if the off-diagonal components are flipped from traditional quadrants. xy should be in the 1st quadaran (0-90 deg) and yx should be in the 3rd quadrant (-180 to -90 deg) :return: A dictionary of components with a bool for flipped or not

if flipped return value is True.

Return type
dict

flip_phase(zxx=False, zxy=False, zyx=False, zyy=False, tzx=False, tzy=False, inplace=False)

Flip the phase of a station in case its plotting in the wrong quadrant. :param inplace:

Defaults to False.

Parameters

- **tzy** – Defaults to False.
- **tzx** – Defaults to False.
- **station (string or list)** – Name(s) of station to flip phase.
- **station** – Station name or list of station names.
- **zxx (TYPE, optional)** – Z_xx, defaults to False.
- **zxy (TYPE, optional)** – Z_xy, defaults to False.
- **zyy (TYPE, optional)** – Z_yx, defaults to False.
- **zyx (TYPE, optional)** – Z_yy, defaults to False.
- **tx (TYPE, optional)** – T_zx, defaults to False.
- **ty (TYPE, optional)** – T_zy, defaults to False.

Returns

New_data.

Return type
np.ndarray

Returns

New mt_dict with components removed.

Return type
dictionary

```
from_dataframe(mt_df, impedance_units='mt')
```

Fill transfer function attributes from a dataframe for a single station. :param mt_df: :param df: DESCRIPTION. :type df: TYPE :param impedance_units: [“mt” [mV/km/nT] | “ohm” [Ohms]] :type impedance_units: str

```
property hx_metadata
```

HX metadata.

```
property hy_metadata
```

HY metadata.

```
property hz_metadata
```

HZ metadata.

```
property impedance_units
```

impedance units

```
interpolate(new_period, method='slinear', bounds_error=True, f_type='period', z_log_space=False, **kwargs)
```

Interpolate the impedance tensor onto different frequencies. :param z_log_space:

Defaults to False.

Parameters

- **new_period** (*np.ndarray*) – A 1-d array of frequencies to interpolate on to. Must be within the bounds of the existing frequency range, anything outside and an error will occur.
- **method** (*string, optional*) – Method to interpolate by, defaults to “slinear”.
- **bounds_error** (*boolean, optional*) – Check for if input frequencies are within the original frequencies, defaults to True.
- **f_type** (*string, defaults to ‘period’, optional*) – Frequency type can be [‘frequency’ | ‘period’], defaults to “period”.
- ****kwargs** – Key word arguments for *interp*.

Raises

ValueError – If input frequencies are out of bounds.

Returns

New MT object with interpolated values.

Return type

`mtpy.core.MT`

```
plot_depth_of_penetration(**kwargs)
```

Plot Depth of Penetration estimated from Niblett-Bostick estimation. :param **kwargs: DESCRIPTION. :type **kwargs: TYPE :return: DESCRIPTION. :rtype: TYPE

```
plot_mt_response(**kwargs)
```

Returns a `mtpy.imaging.plotresponse.PlotResponse` object

Plot Response

```
>>> mt_obj = mt.MT(edi_file)
>>> pr = mt.plot_mt_response()
>>> # if you need more info on plot_mt_response
>>> help(pr).
```

plot_phase_tensor(kwargs)**

Plot phase tensor. :return: DESCRIPTION. :rtype: TYPE

property pt

Mtpy.analysis.pt.PhaseTensor object to hold phase tensor.

remove_component(zxx=False, zxy=False, zyy=False, zyx=False, tzx=False, tzy=False, inplace=False)

Remove a component for a given station(s). :param inplace:

Defaults to False.

Parameters

- **tzy** – Defaults to False.
- **tzx** – Defaults to False.
- **station (string or list)** – Station name or list of station names.
- **zxx (TYPE, optional)** – Z_xx, defaults to False.
- **zxy (TYPE, optional)** – Z_xy, defaults to False.
- **zyy (TYPE, optional)** – Z_yy, defaults to False.
- **zyx (TYPE, optional)** – Z_yx, defaults to False.
- **tx (TYPE, optional)** – T_zx, defaults to False.
- **ty (TYPE, optional)** – T_zy, defaults to False.

Returns

New data array with components removed.

Return type

np.ndarray

Returns

New mt_dict with components removed.

Return type

dictionary

remove_distortion(n_frequencies=None, comp='det', only_2d=False, inplace=False)

Remove distortion following Bibby et al. [2005]. :param inplace:

Defaults to False.

Parameters

- **only_2d** – Defaults to False.
- **comp** – Defaults to “det”.
- **n_frequencies (int, optional)** – Number of frequencies to look for distortion from the highest frequency, defaults to None.

Returns

Distortion matrix.

Rtype s

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

Return s

Z with distortion removed.

Rtype s
mtpy.core.z.Z

remove_static_shift(ss_x=1.0, ss_y=1.0, inplace=False)

Remove static shift from the apparent resistivity

Assume the original observed tensor Z is built by a static shift S and an unperturbed “correct” Z0 :

- $Z = S * Z_0$

therefore the correct Z will be :

- $Z_0 = S^{-1} * Z$.

Parameters

- **inplace** – Defaults to False.
- **ss_x (float, optional)** – Correction factor for x component, defaults to 1.0.
- **ss_y (float, optional)** – Correction factor for y component, defaults to 1.0.

Return s

New Z object with static shift removed.

Rtype s

mtpy.core.z.Z

rotate(theta_r, inplace=True)

Rotate the data in degrees assuming North is 0 measuring clockwise positive to East as 90.

Parameters

- **theta_r (float)** – rotation angle to rotate by in degrees.
- **inplace (bool, optional)** – rotate all transfer function in place, defaults to True.

Returns

if inplace is False, returns a new MT object.

Return type

MT object

property rotation_angle

Rotation angle in degrees from north. In the coordinate reference frame

property rrhx_metadata

RRHX metadata.

property rrhy_metadata

RRHY metadata.

to_dataframe(utm_crs=None, cols=None, impedance_units='mt')

Create a dataframe from the transfer function for use with plotting and modeling. :param cols:

Defaults to None.

Parameters

- **utm_crs (eter)** – Defaults to None.
- **utm_crs** – The utm zone to project station to, could be a name, pyproj.CRS, EPSG number, or anything that pyproj.CRS can intake.

- **impedance_units** (*str*) – [“mt” [mV/km/nT] | “ohm” [Ohms]]

`to_occam1d(data_filename=None, mode='det')`

Write an Occam1DData data file. :param data_filename: Path to write file, if None returns Occam1DData object, defaults to None.

Parameters

`mode (string, optional)` – [‘te’, ‘tm’, ‘det’, ‘tez’, ‘tmz’, ‘detz’], defaults to “det”.

Returns

Occam1DData object.

Return type

`mtpy.modeling.occam1d.Occam1DData`

`to_simpeg_1d(mode='det', **kwargs)`

Helper method to run a 1D inversion using Simpeg

default is smooth parameters

To run sharp inversion

```
>>> mt_object.to_simpeg_1d({"p_s": 2, "p_z": 0, "use_irrls": True})
```

To run sharp inversion adn compact

```
>>> mt_object.to_simpeg_1d({"p_s": 0, "p_z": 0, "use_irrls": True}).  
:param mode:  
    Defaults to "det".  
:param **kwargs: DESCRIPTION.  
:type **kwargs: TYPE  
:return: DESCRIPTION.  
:rtype: TYPE
```

`mtpy.core.mt_collection module`

Collection of MT stations

Created on Mon Jan 11 15:36:38 2021

copyright

Jared Peacock (jpeacock@usgs.gov)

license

MIT

`class mtpy.core.mt_collection.MTCollection(working_directory=None)`

Bases: object

Collection of transfer functions

The main working variable is `MTCollection.dataframe` which is a property that returns either the *master_dataframe* that contains all the TF’s in the MTH5 file, or the `working_dataframe` which is a dataframe that has been queried in some way. Therefore all the user has to do is set the working directory as a subset of the master_dataframe

Example

```
>>> mc = MTCollection()
>>> mc.open_collection(filename="path/to/example/mth5.h5")
>>> mc.working_dataframe = mc.master_dataframe.iloc[0:5].
```

`add_tf(transfer_function, new_survey=None, tf_id_extra=None)`

Transfer_function could be a transfer function object, a file name, a list of either.

Parameters

- **transfer_function** (*list, tuple, array, MTData, MT*) – Transfer function object.
- **new_survey** (*str, optional*) – New survey name, defaults to None.
- **tf_id_extra** (*string, optional*) – Additional text onto existing ‘tf_id’,, defaults to None.

Returns

DESCRIPTION.

Return type

TYPE

`apply_bbox(lon_min: float, lon_max: float, lat_min: float, lat_max: float) → None`

Sets self.working_dataframe to only stations within bounding box.

Parameters

- **lon_min** (*float*) – Minimum longitude.
- **lon_max** (*float*) – Maximum longitude.
- **lat_min** (*float*) – Minimum latitude.
- **lat_max** (*float*) – Maximum longitude.

`average_stations(cell_size_m, bounding_box=None, count=1, n_periods=48, new_file=True)`

Average nearby stations to make it easier to invert.

Parameters

- **new_file** – Defaults to True.
- **n_periods** – Defaults to 48.
- **count** – Defaults to 1.
- **cell_size_m** (*float*) – size of square to look in for nearby stations
- **bounding_box** (*list, optional*) – bounding box [lon_min, lon_max, lat_min, lat_max], defaults to None.

`check_for_duplicates(locate='location', sig_figs=6)`

Check for duplicate station locations in a MT DataFrame.

Parameters

- **sig_figs** – Defaults to 6.
- **locate** – Defaults to “location”.
- **dataframe** (*TYPE*) – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

close_collection()

Close mth5.

Returns

DESCRIPTION.

Return type

TYPE

property dataframe

This property returns the working dataframe or master dataframe if the working dataframe is None.

Returns

DESCRIPTION.

Return type

TYPE

from_mt_data(mt_data, new_survey=None, tf_id_extra=None)

Add data from a MTData object to an MTH5 collection.

Can use ‘new_survey’ to create a new survey to load to.

Can use ‘tf_id_extra’ to add a string onto the existing ‘tf_id’, useful if data have been edited or manipulated in some way. For example could set ‘tf_id_extra’ = ‘rotated’ for rotated data. This will help you organize the tf’s for each station.

Parameters

- **mt_data** ([mtpy.core.mt_data.MTData](#)) – MTData object.
- **new_survey** (str, optional) – New survey name, defaults to None.
- **tf_id_extra** (string, optional) – Additional text onto existing ‘tf_id’,, defaults to None.

Raises**IOError** – If an MTH5 is not writable raises.**get_tf(tf_id, survey=None)**

Get transfer function.

Parameters

- **survey** – Defaults to None.
- **tf_id** (TYPE) – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

has_data()

Has data.

static make_file_list(mt_path, file_types=['edi'])

Get a list of MT file from a given path.

Parameters

- **file_types** – Defaults to [“edi”].
- **mt_path** – Full path to where the MT transfer functions are stored.

property master_dataframe

This is the full summary of all transfer functions in the MTH5 file. It is a property because if a user adds TF's then the master_df will be automatically updated. the transformation is quick for now.

property mth5_filename

Mth5 filename.

open_collection(filename=None, basename=None, working_directory=None, mode='a', **kwargs)

Initialize an mth5.

Parameters

- **mode** – Defaults to “a”.
- **filename** – Defaults to None.
- **basename** (TYPE, optional) – DESCRIPTION, defaults to None.
- **working_directory** (TYPE, optional) – DESCRIPTION, defaults to None.

Returns

DESCRIPTION.

Return type

TYPE

plot_mt_response(tf_id, survey=None, **kwargs)

Plot mt response.

Parameters

- **survey** – Defaults to None.
- **tf_id** (TYPE) – DESCRIPTION.
- ****kwargs** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

plot_penetration_depth_1d(tf_id, survey=None, **kwargs)

Plot 1D penetration depth based on the Niblett-Bostick transformation

Note that data is rotated to estimated strike previous to estimation and strike angles are interpreted for data points that are 3D.

 **See also**

[mtpy.analysis.niblettbostick.calculate_depth_of_investigation](#).

Parameters

- **survey** – Defaults to None.
- **tf_id**

- **tf_object** (*TYPE*) – DESCRIPTION.
- ****kwargs** – DESCRIPTION.

Returns
DESCRIPTION.

Return type
TYPE

plot_penetration_depth_map(*mt_data=None*, ***kwargs*)

Plot Penetration depth in map view for a single period

↳ See also

mtpy.imaging.PlotPenetrationDepthMap.

Parameters

- ****kwargs** –
- **mt_data** (*TYPE*, *optional*) – DESCRIPTION, defaults to None.

Returns
DESCRIPTION.

Return type
TYPE

plot_phase_tensor(*tf_id*, *survey=None*, ***kwargs*)

Plot phase tensor elements.

Parameters

- **survey** – Defaults to None.
- **tf_id** (*TYPE*) – DESCRIPTION.
- ****kwargs** – DESCRIPTION.

Returns
DESCRIPTION.

Return type
TYPE

plot_phase_tensor_map(*mt_data=None*, ***kwargs*)

Plot Phase tensor maps for transfer functions in the working_dataframe

↳ See also

mtpy.imaging.PlotPhaseTensorMaps.

Parameters

- **mt_data** – Defaults to None.
- ****kwargs** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

plot_phase_tensor_pseudosection(*mt_data=None*, ***kwargs*)

Plot a pseudo section of phase tensor ellipses and induction vectors if specified

 **See also**[`mtpy.imaging.PlotPhaseTensorPseudosection`](#)**Parameters**

- **mt_data** – Defaults to None.
- ****kwargs** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

plot_residual_phase_tensor(*mt_data_01*, *mt_data_02*, *plot_type='map'*, ***kwargs*)

Plot residual phase tensor.

Parameters

- **mt_data_01** (*TYPE*) – DESCRIPTION.
- **mt_data_02** (*TYPE*) – DESCRIPTION.
- **plot_type** (*TYPE, optional*) – DESCRIPTION, defaults to “map”.
- ****kwargs** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

plot_resistivity_phase_maps(*mt_data=None*, ***kwargs*)

Plot apparent resistivity and/or impedance phase maps from the working dataframe

 **See also**[`mtpy.imaging.PlotResPhaseMaps`](#)**Parameters**

- **mt_data** – Defaults to None.
- ****kwargs** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

plot_resistivity_phase_pseudosections(*mt_data=None*, ***kwargs*)

Plot resistivity and phase in a pseudosection along a profile line.

Parameters

- ***mt_data*** (*TYPE*, *optional*) – DESCRIPTION, defaults to None.
- *****kwargs*** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

plot_stations(*map_epsg=4326*, *bounding_box=None*, ***kwargs*)

Plot stations.

Parameters

- ***bounding_box*** – Defaults to None.
- ***map_epsg*** – Defaults to 4326.
- *****kwargs*** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

plot_strike(*mt_data=None*, ***kwargs*)

Plot strike angle

**See also**[*mtpy.imaging.PlotStrike*.](#)**to_geo_df**(*bounding_box=None*, *epsg=4326*)

Make a geopandas dataframe for easier GIS manipulation.

to_mt_data(*bounding_box=None*, ***kwargs*)

Get a list of transfer functions.

Parameters

- *****kwargs*** –
- ***tf_ids*** (*TYPE*, *optional*) – DESCRIPTION, defaults to None.
- ***bounding_box*** (*TYPE*, *optional*) – DESCRIPTION, defaults to None.

Returns

DESCRIPTION.

Return type

TYPE

to_shp(*filename*, *bounding_box*=None, *epsg*=4326)

Create a shape file of station locations in the given EPSG number

Parameters

- **filename** (*str*) – filename to save the shape file to.
- **bounding_box** (*list, optional*) – bounding box [lon_min, lon_max, lat_min, lat_max], defaults to None.
- **epsg** (*int, optional*) – EPSG number to write shape file to, defaults to 4326.

Returns

dataframe.

Return type

geopandas.DataFrame

property working_directory

Working directory.

mtpy.core.mt_data module

Created on Mon Oct 10 11:58:56 2022

@author: jpeacock

class mtpy.core.mt_data.MTData(*mt_list*=None, *kwargs*)**

Bases: OrderedDict, *MTStations*

Collection of MT objects as an OrderedDict where keys are formatted as *survey_id.station_id*. Has all functionality of an OrderedDict for example can iterate of .keys(), .values() or .items. Values are a list of MT objects.

Inherits *mtpyt.core.MTStations* to deal with geographic locations of stations.

Is not optimized yet for speed, works fine for smaller surveys, but for large can be slow. Might try using a dataframe as the base.

add_station(*mt_object*, *survey*=None, *compute_relative_location*=True, *interpolate_periods*=None, *compute_model_error*=False)

Add a MT object.

Parameters

- **compute_model_error** – Defaults to False.
- **mt_object** (*mtpy.MT*) – MT object for a single station.
- **survey** (*str, optional*) – New survey name, defaults to None.
- **compute_relative_location** (*bool, optional*) – Compute relative location, can be slow if adding single stations in a loop. If looping over station set to False and compute at the end, defaults to True.
- **interpolate_periods** (*np.array, optional*) – Periods to interpolate onto, defaults to None.

add_tf(*tf*, *kwargs*)**

Add a MT object.

Parameters

- ****kwargs** –

- **tf**
- **mt_object** (*mtpy.MT*) – MT object for a single station.
- **survey** (*str, optional*) – New survey name, defaults to None.
- **compute_relative_location** (*bool, optional*) – Compute relative location, can be slow if adding single stations in a loop. If looping over station set to False and compute at the end, defaults to True.
- **interpolate_periods** (*np.array, optional*) – Periods to interpolate onto, defaults to None.

add_white_noise(*value, inplace=True*)

Add white noise to the data, useful for synthetic tests.

Parameters

- **value** (*TYPE*) – DESCRIPTION.
- **inplace** (*TYPE, optional*) – DESCRIPTION, defaults to True.

Returns

DESCRIPTION.

Return type

TYPE

apply_bounding_box(*lon_min, lon_max, lat_min, lat_max*)

Apply a bounding box.

Parameters

- **lon_min** (*TYPE*) – DESCRIPTION.
- **lon_max** (*TYPE*) – DESCRIPTION.
- **lat_min** (*TYPE*) – DESCRIPTION.
- **lat_max** (*TYPE*) – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

clone_empty()

Return a copy of MTData excluding MT objects.

Returns

Copy of MTData object excluding MT objects.

Return type

mtpy.MTData

compute_model_errors(*z_error_value=None, z_error_type=None, z_floor=None, t_error_value=None, t_error_type=None, t_floor=None*)

Compute mode errors based on the error type

key	definition
egbert	error_value * sqrt(Zxy * Zyx)
geometric_mean	error_value * sqrt(Zxy * Zyx)
arithmetic_mean	error_value * (Zxy + Zyx) / 2
mean_od	error_value * (Zxy + Zyx) / 2
off_diagonals	zxx_err == zxy_err, zyx_err == zyy_err
median	error_value * median(z)
eigen	error_value * mean(eigen(z))
percent	error_value * z
absolute	error_value

Parameters

- **z_error_value** (*TYPE, optional*) – DESCRIPTION, defaults to None.
- **z_error_type** (*TYPE, optional*) – DESCRIPTION, defaults to None.
- **z_floor** (*TYPE, optional*) – DESCRIPTION, defaults to None.
- **t_error_value** (*TYPE, optional*) – DESCRIPTION02, defaults to None.
- **t_error_type** (*TYPE, optional*) – DESCRIPTION, defaults to None.
- **t_floor** (*TYPE, optional*) – DESCRIPTION, defaults to None.

Param

DESCRIPTION.

Type

TYPE

Returns

DESCRIPTION.

Return type

TYPE

property coordinate_reference_frame

coordinate reference frame ned or enu

copy()

Deep copy of original MTData object.

Parameters

memo (*TYPE*) – DESCRIPTION.

Returns

Deep copy of original MTData.

Return type

[mtpy.MTData](#)

estimate_spatial_static_shift(station_key, radius, period_min, period_max, radius_units='m', shift_tolerance=0.15)

Estimate static shift for a station by estimating the median resistivity values for nearby stations within a radius given. Can set the period range to estimate the resistivity values.

Parameters

- **shift_tolerance** – Defaults to 0.15.

- **radius_units** – Defaults to “m”.
- **station_key** (TYPE) – DESCRIPTION.
- **radius** (TYPE) – DESCRIPTION.
- **period_min** (TYPE) – DESCRIPTION.
- **period_max** (TYPE) – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

estimate_starting_rho()

Estimate starting resistivity from the data.

Creates a plot of the mean and median apparent resistivity values.

Returns

Array of the median rho per period.

Return type

np.ndarray(n_periods)

Returns

Array of the mean rho per period.

Return type

np.ndarray(n_periods)

from_dataframe(df, impedance_units='mt')

Create an dictionary of MT objects from a dataframe.

Parameters

- **df** (pandas.DataFrame) – Dataframe of mt data.
- **impedance_units** (str) – [“mt” [mV/km/nT] | “ohm” [Ohms]]

from_modem(data_filename, survey='data', **kwargs)

Read in a modem data file

Parameters

- **survey** – Defaults to “data”.
- **data_filename** (TYPE) – DESCRIPTION.
- ****kwargs** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

from_modem_data(data_filename, survey='data', **kwargs)

From modem data.

Parameters

- **survey** – Defaults to “data”.

- **data_filename** (*TYPE*) – DESCRIPTION.
- **file_type** (*TYPE, optional*) – DESCRIPTION, defaults to “data”.
- ****kwargs** – DESCRIPTION.

Returns

DESCRIPTION.

Return type*TYPE***from_mt_dataframe**(*mt_df, impedance_units='mt'*)

Create an dictionary of MT objects from a dataframe.

Parameters

- **mt_df**
- **df** (*MTDataFrame*) – Dataframe of mt data.
- **impedance_units** (*str*) – [“mt” [mV/km/nT] | “ohm” [Ohms]]

from_occam2d(*data_filename, file_type='data', **kwargs*)

Read in occam data from a 2D data file *.dat

Read data file and plot

```
>>> from mtpy import MTData
>>> md = MTData()
>>> md.from_occam2d_data(f"/path/to/data/file.dat")
>>> plot_stations = md.plot_stations(model_locations=True)
```

Read response file

```
>>> md.from_occam2d_data(f"/path/to/response/file.dat")
```

Note

When reading in a response file the survey will be called model. So now you can have the data and model response in the same object..

from_occam2d_data(*data_filename, file_type='data', **kwargs*)

From occam2d data.

get_nearby_stations(*station_key, radius, radius_units='m'*)

Get stations close to a given station.

Parameters

- **radius_units** – Defaults to “m”.
- **station_key** (*TYPE*) – DESCRIPTION.
- **radius** (*TYPE*) – DESCRIPTION.

Returns

DESCRIPTION.

Return type*TYPE*

get_periods()

Get all unique periods.

Returns

DESCRIPTION.

Return type

TYPE

get_profile(*x1, y1, x2, y2, radius*)

Get stations along a profile line given the (x1, y1) and (x2, y2) coordinates within a given radius (in meters).

These can be in (longitude, latitude) or (easting, northing). The calculation is done in UTM, therefore a UTM CRS must be input

Parameters

- **x1** (TYPE) – DESCRIPTION.
- **y1** (TYPE) – DESCRIPTION.
- **x2** (TYPE) – DESCRIPTION.
- **y2** (TYPE) – DESCRIPTION.
- **radius** (TYPE) – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

get_station(*station_id=None, survey_id=None, station_key=None*)

If ‘station_key’ is None, tries to find key from *station_id* and ‘survey_id’ using MTData._get_station_key()

Parameters

- **station_key** (str, optional) – Full station key {survey_id}.{station_id},, defaults to None.
- **station_id** (str, optional) – Station ID, defaults to None.
- **survey_id** (str, optional) – Survey ID, defaults to None.

Raises

KeyError – If cannot find station_key.

Returns

MT object.

Return type

[mtpy.MT](#)

get_subset(*station_list*)

Get a subset of the data from a list of stations, could be station_id or station_keys. Safest to use keys {survey}.{station}

Parameters

station_list (list) – List of station keys as {survey_id}.{station_id}.

Returns

Returns just those stations within station_list.

Return type

`mtpy.MTData`

get_survey(*survey_id*)

Get all MT objects that belong to the ‘survey_id’ from the data set.

Parameters

`survey_id (str)` – Survey ID.

Returns

MTData object including only those with the desired ‘survey_id’.

Return type

`mtpy.MTData`

property impedance_units

impedance units

interpolate(*new_periods*, *f_type='period'*, *inplace=True*, *bounds_error=True*, *kwargs*)**

Interpolate onto common period range

kwargs include

- method
- bounds_error
- z_log_space
- na_method
- extrapolate

Parameters

- `new_periods` (`TYPE`) – DESCRIPTION
- `inplace` – Defaults to True.
- `new_periods` – DESCRIPTION.
- `f_type` (`string, defaults to 'period', optional`) – Frequency type can be [‘frequency’ | ‘period’], defaults to “period”.

Returns

DESCRIPTION.

Return type

`TYPE`

property mt_list

Mt list.

Returns

List of MT objects.

Return type

list

property n_stations

Number of stations in MT data.

plot_mt_response(*station_key=None*, *station_id=None*, *survey_id=None*, ***kwargs*)

Plot mt response.

Parameters

- **survey_id** – Defaults to None.
- **station_id** – Defaults to None.
- **station_key** – Defaults to None.
- **tf_id** (*TYPE*) – DESCRIPTION.
- ****kwargs** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

plot_penetration_depth_1d(*station_key=None*, *station_id=None*, *survey_id=None*, ***kwargs*)

Plot 1D penetration depth based on the Niblett-Bostick transformation

Note that data is rotated to estimated strike previous to estimation and strike angles are interpreted for data points that are 3D.

↳ **See also**

[mtpy.analysis.niblettbostick.calculate_depth_of_investigation](#).

Parameters

- **survey_id** – Defaults to None.
- **station_id** – Defaults to None.
- **station_key** – Defaults to None.
- **tf_object** (*TYPE*) – DESCRIPTION.
- ****kwargs** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

plot_penetration_depth_map(***kwargs*)

Plot Penetration depth in map view for a single period

↳ **See also**

[mtpy.imaging.PlotPenetrationDepthMap](#).

Parameters

- ****kwargs** –
- **mt_data** (*TYPE*) – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

plot_phase_tensor(*station_key=None*, *station_id=None*, *survey_id=None*, ***kwargs*)

Plot phase tensor elements.

Parameters

- **survey_id** – Defaults to None.
- **station_id** – Defaults to None.
- **station_key** – Defaults to None.
- **tf_id** (TYPE) – DESCRIPTION.
- ****kwargs** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

plot_phase_tensor_map(***kwargs*)

Plot Phase tensor maps for transfer functions in the working_dataframe

**See also**[*mtpy.imaging.PlotPhaseTensorMaps*](#).**Parameters******kwargs** – DESCRIPTION.**Returns**

DESCRIPTION.

Return type

TYPE

plot_phase_tensor_pseudosection(*mt_data=None*, ***kwargs*)

Plot a pseudo section of phase tensor ellipses and induction vectors if specified

**See also**[*mtpy.imaging.PlotPhaseTensorPseudosection*](#)**Parameters**

- **mt_data** – Defaults to None.
- ****kwargs** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

plot_residual_phase_tensor_maps(*survey_01*, *survey_02*, ***kwargs*)

Plot residual phase tensor maps.

Parameters

- ****kwargs** –
- **survey_01** (TYPE) – DESCRIPTION.
- **survey_02** (TYPE) – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

plot_resistivity_phase_maps(***kwargs*)

Plot apparent resistivity and/or impedance phase maps from the working dataframe

 See also[mtpy.imaging.PlotResPhaseMaps](#)**Parameters******kwargs** – DESCRIPTION.**Returns**

DESCRIPTION.

Return type

TYPE

plot_resistivity_phase_pseudosections(***kwargs*)

Plot resistivity and phase in a pseudosection along a profile line.

Parameters

- **mt_data** (TYPE, optional) – DESCRIPTION, defaults to None.
- ****kwargs** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

plot_stations(*map_epsg*=4326, *bounding_box*=None, *model_locations*=False, ***kwargs*)

Plot stations.

Parameters

- **model_locations** – Defaults to False.
- **bounding_box** – Defaults to None.
- **map_epsg** – Defaults to 4326.

- ****kwargs** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

plot_strike(kwargs)**

Plot strike angle

 **See also**

[`mtpy.imaging.PlotStrike.`](#)

plot_tipper_map(kwargs)**

Plot Phase tensor maps for transfer functions in the working_dataframe

 **See also**

[`mtpy.imaging.PlotPhaseTensorMaps.`](#)

Parameters

- ****kwargs** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

remove_station(station_id, survey_id=None)

Remove a station from the dictionary based on the key.

Parameters

- **station_id** (*str*) – Station ID.
- **survey_id** (*str, optional*) – Survey ID, defaults to None.

rotate(rotation_angle, inplace=True)

Rotate the data by the given angle assuming positive clockwise with north = 0, east = 90.

Parameters

- **inplace** – Defaults to True.
- **rotation_angle** (*TYPE*) – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

property survey_ids

Survey IDs for all MT objects.

Returns

List of survey IDs.

Return type

list

to_dataframe(*utm_crs=None*, *cols=None*, *impedance_units='mt'*)

To dataframe.

Parameters

- **utm_crs** (*TYPE*, *optional*) – DESCRIPTION, defaults to None.
- **cols** (*TYPE*, *optional*) – DESCRIPTION, defaults to None.
- **impedance_units** (*str*) – [“mt” [mV/km/nT] | “ohm” [Ohms]]

Returns

DESCRIPTION.

Return type

TYPE

to_geo_df(*model_locations=False*, *data_type='station_locations'*)

Make a geopandas dataframe for easier GIS manipulation.

Parameters

- **model_locations** (*bool*, *optional*) – If True returns points in model coordinates,, defaults to False.
- **data_type** (*string*, *optional*) – Type of data in GeoDataFrame [‘station_locations’ | ‘phase_tensor’ | ‘tipper’ | ‘shapefiles’ (both pt and tipper)], defaults to “station_locations”.

Returns

Geopandas dataframe with requested data in requested coordinates.

Return type

geopandas.GeoDataFrame

to_modem(*data_filename=None*, ***kwargs*)

Create a modem data file.

Parameters

- **data_filename** (*TYPE*, *optional*) – DESCRIPTION, defaults to None.
- ****kwargs** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

to_modem_data(*data_filename=None*, ***kwargs*)

To modem data.

to_mt_dataframe(*utm_crs=None*, *impedance_units='mt'*)

Create an MTDataFrame.

Parameters

- **utm_crs** (*TYPE, optional*) – DESCRIPTION, defaults to None.
- **impedance_units** (*str*) – [“mt” [mV/km/nT] | “ohm” [Ohms]]

Returns

DESCRIPTION.

Return type

TYPE

to_occam2d(*data_filename=None, **kwargs*)

Write an Occam2D data file.

Parameters

- **data_filename** (*TYPE, optional*) – DESCRIPTION, defaults to None.
- ****kwargs** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

to_occam2d_data(*data_filename=None, **kwargs*)

To occam2d data.

to_shp_pt_tipper(*save_dir, output_crs=None, utm=False, pt=True, tipper=True, periods=None, period_tol=None, ellipse_size=None, arrow_size=None*)

Write phase tensor and tipper shape files.

Note

If you have a mixed data set such that the periods do not match up, you should first interpolate onto a common period map and then make shape files. Otherwise you will have a bunch of shapefiles with only a few shapes.

Parameters

- **arrow_size** – Defaults to None.
- **ellipse_size** – Defaults to None.
- **utm** – Defaults to False.
- **save_dir** (*string or Path*) – Folder to save shape files to.
- **output_crs** (*string, CRS, optional*) – CRS the output shape files will be in, depending of if utm is True or False, defaults to None.
- **pt** (*bool, optional*) – Make phase tensor shape files, defaults to True.
- **tipper** (*bool, optional*) – Make tipper shape files, defaults to True.
- **periods** (*np.ndarray, optional*) – Periods to plot periods within the data, defaults to None.
- **period_tol** (*float, optional*) – Tolerance to search around periods, defaults to None.

Returns

Dictionary of file paths.

Return type
dictionary

`to_simpeg_2d(**kwargs)`

Create a data object for Simpeg to work with.

All information is derived from the dataframe. Therefore the user should create the profile, interpolate, estimate model errors from the *MTData* object first before creating the Simpeg2D object.

kwargs include:

- *include_elevation* -> bool
- *invert_te* -> bool
- *invert_tm* -> bool

`to_simpeg_3d(**kwargs)`

Create a data object that Simpeg can work with.

All information is derived from the dataframe. Therefore the user should interpolate, estimate model errors, etc from the *MTData* object first before creating the Simpeg3D object.

kwargs include:

- *include_elevation* -> bool
- *geographic_coordinates* -> bool
- *invert_z_xx* -> bool
- *invert_z_xy* -> bool
- *invert_z_yx* -> bool
- *invert_z_yy* -> bool
- *invert_t_zx* -> bool
- *invert_t_zy* -> bool
- *invert_types* = [“real”, “imaginary”]

`mtpy.core.mt_dataframe module`

Created on Sun Oct 2 13:20:28 2022

@author: jpeacock

`class mtpy.core.mt_dataframe.MTDataFrame(data=None, n_entries=0, **kwargs)`

Bases: `object`

Dataframe for a single station

Tried subclassing pandas.DataFrame, but that turned out to not be straight forward, so went with compilation instead.

Think about having period as an index?.

`property datum_epsg`

Datum_epsg.

`property east`

Station.

property elevation
Elevation.

property for_shapefiles
For shape files includes phase tensor and tippe.

property frequency
Get frequencies. :return: DESCRIPTION. :rtype: TYPE

from_t_object(t_object)
Fill tipper. :param t_object: :param tipper: DESCRIPTION. :type tipper: TYPE :param tipper_error: DESCRIPTION. :type tipper_error: TYPE :param index: DESCRIPTION. :type index: TYPE :return: DESCRIPTION. :rtype: TYPE

from_z_object(z_object)
Fill impedance. :param z_object: :param impedance: DESCRIPTION. :type impedance: TYPE :param index: DESCRIPTION. :type index: TYPE :return: DESCRIPTION. :rtype: TYPE

get_period(period, tol=None)
Get periods with a percentage based on tol if given. :param period: Exact period value to search for. :type period: float :param tol: Tolerance to search around given period, defaults to None. :type tol: float, optional :return: Dataframe with periods. :rtype: TYPE

get_station_df(station=None)
Get a single station df. :return: DESCRIPTION. :rtype: TYPE

get_station_distances(utm=False)
Get distance information between stations. :return: DESCRIPTION. :rtype: TYPE

property impedance
Impedance elements.

property latitude
Latitude.

property longitude
Longitude.

property model_east
Model_east.

property model_elevation
Model_elevation.

property model_north
Model_north.

property nonzero_items
Return number of non zero entries.

property north
North.

property period
Get frequencies. :return: DESCRIPTION. :rtype: TYPE

property phase_tensor
Phase tensor information.

```
property profile_offset
    Profile_offset.

property size
    Size function.

property station
    Station name.

property station_locations
    Station locations.

property survey
    Survey name.

property tipper
    Phase tensor information.

to_t_object()
    To a tipper object. :return: DESCRIPTION. :rtype: TYPE

to_z_object(units='mt')
    Fill z_object from dataframe
    Need to have the components this way for transposing the elements so that the shape is (nf, 2, 2).

property utm_epsg
    Utm_epsg.
```

mtpy.core.mt_location module

Might think about adding declination

Created on Mon Oct 3 15:04:12 2022

@author: jpeacock

```
class mtpy.core.mt_location.MTLocation(survey_metadata=None, **kwargs)
    Bases: object
    Location for a MT site or point measurement.

compute_model_location(center_location)
    Compute model location based on model center and model epsg. :param center_location: :param
    model_center: DESCRIPTION. :type model_center: TYPE :return: DESCRIPTION. :rtype: TYPE

copy()
    Copy function.

property datum_crs
    Datum crs.

property datum_epsg
    Datum epsg.

property datum_name
    Datum name.
```

```
property east
    Easting.

property elevation
    Elevation function.

from_json(filename)
    Read in json formatted location. :param filename: DESCRIPTION. :type filename: TYPE :return: DESCRIPTION. :rtype: TYPE

get_elevation_from_national_map()
    Get elevation from DEM data of the US National Map.

    Plan to extend
    this to the globe.

    Pulls data from the USGS national map DEM :return: DESCRIPTION. :rtype: TYPE

property latitude
    Latitude function.

property longitude
    Longitude function.

property model_east
    Model east.

property model_elevation
    Model elevation.

property model_north
    Model north.

property north
    Northing.

project_onto_profile_line(profile_slope, profile_intersection)
    Project onto profile line. :param profile_slope: DESCRIPTION. :type profile_slope: TYPE :param profile_intersection: DESCRIPTION. :type profile_intersection: TYPE :param units: DESCRIPTION, defaults to "deg". :type units: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

to_json(filename)
    Write out information to a json file. :param filename: DESCRIPTION. :type filename: TYPE :return: DESCRIPTION. :rtype: TYPE

property utm_crs
    Utm crs.

property utm_epsg
    Utm epsg.

property utm_name
    Utm name.

property utm_zone
    Utm zone.
```

mtpy.core.mt_stations module**ModEM**

```
# Generate files for ModEM
# revised by JP 2017 # revised by AK 2017 to bring across functionality from ak branch
class mtpy.core.mt_stations.MTStations(utm_epsg, datum_epsg=None, **kwargs)
Bases: object

Object to deal with station location and geographic projection.

Geographic projections are done using pyproj.CRS objects.

Takes in a list of mtpy.core.mt.MT objects which are inherit mtpy.core.mt\_location.MTLocation objects,
which deal with transformation of point data using pyproj.
```

property center_point

Calculate the center point from the given station locations

If _center attributes are set, that is returned as the center point.

Otherwise, looks for non-zero locations in E-N first, then Lat/Lon and estimates the center point as (max - min) / 2...

Returns

Center point.

Return type

[mtpy.core.MTLocation](#)

center_stations(model_obj)

Center station locations to the middle of cells, is useful for topography cause it reduces edge effects of stations close to cell edges. Recalculates rel_east, rel_north to center of model cell.

Parameters

model_obj ([mtpy.modeling.modem.Model](#)) – mtpy.modeling.Structured object of the model.

compute_relative_locations()

Calculate model station locations relative to the center point in meters.

Uses [mtpy.core.MTLocation.compute_model_location](#) to calculate the relative distance.

Computes inplace.

copy()

Create a deep copy of the MTStations object.

Note

At the moment this is very slow because it is making a lot of deep copies. Use sparingly.

Returns

Deep copy of MTStation object.

Return type

[mtpy.core.mt_stations.MTStations](#)

property datum_crs

Datum crs.

Returns

Datum CRS object.

Return type

`pypyproj.CRS`

property datum_epsg

Datum epsg.

Returns

Datum EPSG number.

Return type

`int`

property datum_name

Datum name.

Returns

Datum well known name.

Return type

`str`

generate_profile(units='deg')

Estimate a profile from the data.

Returns

DESCRIPTION.

Return type

`TYPE`

generate_profile_from_strike(strike, units='deg')

Estimate a profile line from a given geoelectric strike.

Parameters

- **strike**
- **units** (`TYPE`, *optional*) – DESCRIPTION. By default, “deg”.

Returns

DESCRIPTION.

Return type

`TYPE`

property model_epsg

Model epsg.

Returns

Model epsg number from the model_crs object.

Return type

`int`

project_stations_on_topography(model_object, air_resistivity=1000000000000.0, sea_resistivity=0.3, ocean_bottom=False)

Project stations on topography of a given model.

Parameters

- **model_object**
- **model_obj** (*mtpy.modeling.modem.Model*) – *mtpy.modeling.modem.Model* object of the model.
- **air_resistivity** (*float, optional*) – Resistivity value of air cells in the model. By default, 1e12.
- **sea_resistivity** (*float, optional*) – Resistivity of sea3. By default, 0.3.
- **ocean_bottom** (*boolean, optional*) – If True places stations at bottom of sea cells. By default, False.

rotate_stations(*rotation_angle*)

Rotate stations model postions only assuming N is 0 and east is 90.

Note

Computes in place and rotates according to already set rotation angle. Therefore if the station locations have already been rotated the function will rotate the already rotated stations. For example if you rotate the stations 15 degrees, then again by 20 degrees the resulting station locations will be 35 degrees rotated from the original locations.

Parameters

- rotation_angle** (*float*) – Rotation angle in degrees assuming N=0, E=90. Positive clockwise.

property station_locations

Station locations.

Returns

Dataframe of station location information.

Return type

`pandas.DataFrame`

to_csv(*csv_fn, geometry=False*)

Write a shape file of the station locations using geopandas which only takes in epsg numbers

Parameters

- **geometry** – By default, False.
- **csv_fn** (*string*) – Full path to new shapefile.

to_geopd()

Create a geopandas dataframe.

Returns

Geopandas DataFrame with points from latitude and longitude.

Return type

`geopandas.DataFrame`

to_shp(*shp_fn*)

Write a shape file of the station locations using geopandas which only takes in epsg numbers

Parameters

shp_fn (*string*) – Full path to new shapefile.

to_vtk(*vtk_fn=None*, *vtk_save_path=None*, *vtk_fn_basename='ModEM_stations'*, *geographic=False*, *shift_east=0*, *shift_north=0*, *shift_elev=0*, *units='km'*, *coordinate_system='nez+'*)

Write a VTK file for plotting in 3D like Paraview.

Parameters

- **coordinate_system** – By default, “nez”.
- **units** – By default, “km”.
- **shift_elev** – By default, 0.
- **shift_north** – By default, 0.
- **shift_east** – By default, 0.
- **geographic** – By default, False.
- **vtk_fn** (*string or Path, optional*) – Full path to VTK file to be written. By default, None.
- **vtk_save_path** (*string or Path, optional*) – Directory to save vtk file to. By default, None.
- **vtk_fn_basename** – Filename basename of vtk file, note that .vtr. By default, “ModEM_stations”.

Returns

Full path to VTK file.

Return type

Path

property utm_crs

Utm crs.

Returns

UTM CRS object.

Return type

`pypyproj.CRS`

property utm_epsg

Utm epsg.

Returns

UTM EPSG number.

Return type

int

property utm_name

Utm name.

Returns

UTM CRS name.

Return type

string

property utm_zone

Utm zone.

Returns

UTM Zone number.

Return type

str

Module contents**class mtpy.core.MTDataFrame(data=None, n_entries=0, **kwargs)**

Bases: object

Dataframe for a single station

Tried subclassing pandas.DataFrame, but that turned out to not be straight forward, so when with compilation instead.

Think about having period as an index?.

property datum_epsg

Datum_epsg.

property east

Station.

property elevation

Elevation.

property for_shapefiles

For shape files includes phase tensor and tippe.

property frequency

Get frequencies. :return: DESCRIPTION. :rtype: TYPE

from_t_object(t_object)

Fill tipper. :param t_object: :param tipper: DESCRIPTION. :type tipper: TYPE :param tipper_error: DESCRIPTION. :type tipper_error: TYPE :param index: DESCRIPTION. :type index: TYPE :return: DESCRIPTION. :rtype: TYPE

from_z_object(z_object)

Fill impedance. :param z_object: :param impedance: DESCRIPTION. :type impedance: TYPE :param index: DESCRIPTION. :type index: TYPE :return: DESCRIPTION. :rtype: TYPE

get_period(period, tol=None)

Get periods with a percentage based on tol if given. :param period: Exact period value to search for. :type period: float :param tol: Tolerance to search around given period, defaults to None. :type tol: float, optional :return: Dataframe with periods. :rtype: TYPE

get_station_df(station=None)

Get a single station df. :return: DESCRIPTION. :rtype: TYPE

get_station_distances(utm=False)

Get distance information between stations. :return: DESCRIPTION. :rtype: TYPE

property impedance

Impedance elements.

```
property latitude
    Latitude.

property longitude
    Longitude.

property model_east
    Model_east.

property model_elevation
    Model_elevation.

property model_north
    Model_north.

property nonzero_items
    Return number of non zero entries.

property north
    North.

property period
    Get frequencies. :return: DESCRIPTION. :rtype: TYPE

property phase_tensor
    Phase tensor information.

property profile_offset
    Profile_offset.

property size
    Size function.

property station
    Station name.

property station_locations
    Station locations.

property survey
    Survey name.

property tipper
    Phase tensor information.

to_t_object()
    To a tipper object. :return: DESCRIPTION. :rtype: TYPE

to_z_object(units='mt')
    Fill z_object from dataframe

    Need to have the components this way for transposing the elements so that the shape is (nf, 2, 2).

property utm_epsg
    Utm_epsg.

class mtpy.core.MTLocation(survey_metadata=None, **kwargs)
    Bases: object

    Location for a MT site or point measurement.
```

compute_model_location(*center_location*)

Compute model location based on model center and model epsg. :param center_location: :param model_center: DESCRIPTION. :type model_center: TYPE :return: DESCRIPTION. :rtype: TYPE

copy()

Copy function.

property datum_crs

Datum crs.

property datum_epsg

Datum epsg.

property datum_name

Datum name.

property east

Easting.

property elevation

Elevation function.

from_json(*filename*)

Read in json formatted location. :param filename: DESCRIPTION. :type filename: TYPE :return: DESCRIPTION. :rtype: TYPE

get_elevation_from_national_map()

Get elevation from DEM data of the US National Map.

Plan to extend

this to the globe.

Pulls data from the USGS national map DEM :return: DESCRIPTION. :rtype: TYPE

property latitude

Latitude function.

property longitude

Longitude function.

property model_east

Model east.

property model_elevation

Model elevation.

property model_north

Model north.

property north

Northing.

project_onto_profile_line(*profile_slope*, *profile_intersection*)

Project onto profile line. :param profile_slope: DESCRIPTION. :type profile_slope: TYPE :param profile_intersection: DESCRIPTION. :type profile_intersection: TYPE :param units: DESCRIPTION, defaults to “deg”. :type units: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

to_json(*filename*)

Write out information to a json file. :param filename: DESCRIPTION. :type filename: TYPE :return: DESCRIPTION. :rtype: TYPE

property utm_crs

Utm crs.

property utm_epsg

Utm epsg.

property utm_name

Utm name.

property utm_zone

Utm zone.

class mtpy.core.MTStations(*utm_epsg*, *datum_epsg=None*, *kwargs*)**

Bases: object

Object to deal with station location and geographic projection.

Geographic projections are done using pyproj.CRS objects.

Takes in a list of [mtpy.core.mt.MT](#) objects which are inherit [mtpy.core.mt_location.MTLocation](#) objects, which deal with transformation of point data using pyproj.

property center_point

Calculate the center point from the given station locations

If _center attributes are set, that is returned as the center point.

Otherwise, looks for non-zero locations in E-N first, then Lat/Lon and estimates the center point as (max - min) / 2...

Returns

Center point.

Return type

[mtpy.core.MTLocation](#)

center_stations(*model_obj*)

Center station locations to the middle of cells, is useful for topography cause it reduces edge effects of stations close to cell edges. Recalculates rel_east, rel_north to center of model cell.

Parameters

model_obj ([mtpy.modeling.modem.Model](#)) – mtpy.modeling.Structured object of the model.

compute_relative_locations()

Calculate model station locations relative to the center point in meters.

Uses [mtpy.core.MTLocation.compute_model_location](#) to calculate the relative distance.

Computes inplace.

copy()

Create a deep copy of the MTStations object.

Note

At the moment this is very slow because it is making a lot of deep copies. Use sparingly.

Returns

Deep copy of MTStation object.

Return type

`mtpy.core.mt_stations.MTStations`

property datum_crs

Datum crs.

Returns

Datum CRS object.

Return type

`pypyproj.CRS`

property datum_epsg

Datum epsg.

Returns

Datum EPSG number.

Return type

`int`

property datum_name

Datum name.

Returns

Datum well known name.

Return type

`str`

generate_profile(units='deg')

Estimate a profile from the data.

Returns

`DESCRIPTION`.

Return type

`TYPE`

generate_profile_from_strike(strike, units='deg')

Estimate a profile line from a given geoelectric strike.

Parameters

- **strike**
- **units** (`TYPE`, optional) – `DESCRIPTION`. By default, “deg”.

Returns

`DESCRIPTION`.

Return type

`TYPE`

property model_epsg

Model epsg.

Returns

Model epsg number from the model_crs object.

Return type

int

project_stations_on_topography(model_object, air_resistivity=1000000000000.0, sea_resistivity=0.3, ocean_bottom=False)

Project stations on topography of a given model.

Parameters

- **model_object**
- **model_obj** (*mtpy.modeling.modem.Model*) – *mtpy.modeling.modem.Model* object of the model.
- **air_resistivity** (*float, optional*) – Resistivity value of air cells in the model. By default, 1e12.
- **sea_resistivity** (*float, optional*) – Resistivity of sea3. By default, 0.3.
- **ocean_bottom** (*boolean, optional*) – If True places stations at bottom of sea cells. By default, False.

rotate_stations(rotation_angle)

Rotate stations model postions only assuming N is 0 and east is 90.

Note

Computes in place and rotates according to already set rotation angle. Therefore if the station locations have already been rotated the function will rotate the already rotated stations. For example if you rotate the stations 15 degrees, then again by 20 degrees the resulting station locations will be 35 degrees rotated from the original locations.

Parameters

- rotation_angle** (*float*) – Rotation angle in degrees assuming N=0, E=90. Positive clockwise.

property station_locations

Station locations.

Returns

Dataframe of station location information.

Return type

pandas.DataFrame

to_csv(csv_fn, geometry=False)

Write a shape file of the station locations using geopandas which only takes in epsg numbers

Parameters

- **geometry** – By default, False.
- **csv_fn** (*string*) – Full path to new shapefile.

to_geopd()

Create a geopandas dataframe.

Returns

Geopandas DataFrame with points from latitude and longitude.

Return type

geopandas.DataFrame

to_shp(shp_fn)

Write a shape file of the station locations using geopandas which only takes in epsg numbers

Parameters

shp_fn (*string*) – Full path to new shapefile.

to_vtk(vtk_fn=None, vtk_save_path=None, vtk_fn_basename='ModEM_stations', geographic=False, shift_east=0, shift_north=0, shift_elev=0, units='km', coordinate_system='nez+')

Write a VTK file for plotting in 3D like Paraview.

Parameters

- **coordinate_system** – By default, “nez”.
- **units** – By default, “km”.
- **shift_elev** – By default, 0.
- **shift_north** – By default, 0.
- **shift_east** – By default, 0.
- **geographic** – By default, False.
- **vtk_fn** (*string or Path, optional*) – Full path to VTK file to be written. By default, None.
- **vtk_save_path** (*string or Path, optional*) – Directory to save vtk file to. By default, None.
- **vtk_fn_basename** – Filename basename of vtk file, note that .vtr. By default, “ModEM_stations”.

Returns

Full path to VTK file.

Return type

Path

property utm_crs

Utm crs.

Returns

UTM CRS object.

Return type

pypyproj.CRS

property utm_epsg

Utm epsg.

Returns

UTM EPSG number.

Return type

int

property utm_name

Utm name.

Returns

UTM CRS name.

Return type

string

property utm_zone

Utm zone.

Returns

UTM Zone number.

Return type

str

class mtpy.core.PhaseTensor(*z=None, z_error=None, z_model_error=None, frequency=None, pt=None, pt_error=None, pt_model_error=None*)Bases: [TFBase](#)

PhaseTensor class - generates a Phase Tensor (phase tensor) object.

Methods include reading and writing from and to edi-objects, rotations combinations of Z instances, as well as calculation of invariants, inverse, amplitude/phase,...

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

- phase tensor_xx: (0,0)
- phase tensor_xy: (0,1)
- phase tensor_yx: (1,0)
- phase tensor_yy: (1,1)

All internal methods are based on (Caldwell et al.,2004) and (Bibby et al.,2005), in which they use the canonical cartesian 2D reference (x1, x2). However, all components, coordinates, and angles for in- and outputs are given in the geographical reference frame:

- x-axis = North
- y-axis = East
- z-axis = Down

Therefore, all results from using those methods are consistent (angles are referenced from North rather than x1).

property alpha

Principal axis angle (strike) of phase tensor in degrees.

property alpha_error

Principal axis angle error of phase tensor in degrees.

property alpha_model_error

Principal axis angle model error of phase tensor in degrees.

property azimuth

Azimuth angle related to geoelectric strike in degrees.

property azimuth_error

Azimuth angle error related to geoelectric strike in degrees.

property azimuth_model_error

Azimuth angle model error related to geoelectric strike in degrees.

property beta

3D-dimensionality angle Beta (invariant) of phase tensor in degrees.

property beta_error

3D-dimensionality angle error Beta of phase tensor in degrees.

property beta_model_error

3D-dimensionality angle model error Beta of phase tensor in degrees.

property det

Determinant of phase tenso.

property det_error

Determinant error of phase tenso.

property det_model_error

Determinant model erro of phase tenso.

property eccentricity

Eccentricity estimation of dimensionality.

property eccentricity_error

Error in eccentricity estimation.

property eccentricity_model_error

Error in eccentricity estimation.

property ellipticity

Ellipticity of the phase tensor, related to dimesionality.

property ellipticity_error

Ellipticity error of the phase tensor, related to dimesionality.

property ellipticity_model_error

Ellipticity model error of the phase tensor, related to dimesionality.

property only1d

Return phase tensor in 1D form.

If phase tensor 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 phase tensor off-diagonal absolutes.

property only2d

Return phase tensor in 2D form.

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

property phimax

Maximum phase calculated according to Bibby et al. 2005:

Phi_max = Pi2 + Pi1.

```
property phimax_error
    Maximum phase erro.

property phimax_model_error
    Maximum phase model erro.

property phimin
    Minimum phase calculated according to Bibby et al. 2005:
    Phi_min = Pi2 - Pi1.

property phimin_error
    Minimum phase erro.

property phimin_model_error
    Minimum phase model erro.

property pt
    Phase tensor array.

property pt_error
    Phase tensor erro.

property pt_model_error
    Phase tensor model erro.

property skew
    3D-dimensionality skew angle of phase tensor in degrees.

property skew_error
    3D-dimensionality skew angle error of phase tensor in degrees.

property skew_model_error
    3D-dimensionality skew angle model error of phase tensor in degrees.

property trace
    Trace of phase tenso.

property trace_error
    Trace error of phase tenso.

property trace_model_error
    Trace model error of phase tenso.

class mtpy.core.Tipper(tipper=None, tipper_error=None, frequency=None, tipper_model_error=None)
Bases: TFBase

Tipper class -> generates a Tipper-object.

Errors are given as standard deviations (sqrt(VAR)) :param tipper: Tipper array in the shape of [Tx, Ty]
    default is None.
```

Parameters

- **tipper_error** (`np.ndarray((nf, 1, 2))`) – Array of estimated tipper errors in the shape of [Tx, Ty]. Must be the same shape as tipper. *default* is None.
- **frequency** (`np.ndarray(nf)`) – Array of frequencyencies corresponding to the tipper elements. Must be same length as tipper. *default* is None.

property amplitude

Amplitude function.

property amplitude_error

Amplitude error.

property amplitude_model_error

Amplitude model error.

property angle_error

Angle error.

property angle_imag

Angle imag.

property angle_model_error

Angle model error.

property angle_real

Angle real.

property mag_error

Mag error.

property mag_imag

Mag imag.

property mag_model_error

Mag model error.

property mag_real

Mag real.

property phase

Phase function.

property phase_error

Phase error.

property phase_model_error

Phase model error.

set_amp_phase(r, ϕ)

Set values for amplitude(r) and argument (ϕ - in degrees).

Updates the attributes:

- tipper
- tipper_error

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

property tipper

Tipper function.

property tipper_error

Tipper error.

property tipper_model_error

Tipper model error.

class mtpy.core.Z(z=None, z_error=None, frequency=None, z_model_error=None, units='mt')

Bases: [TFBase](#)

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

Z is a complex array of the form (n_frequency, 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)) :param z: Array containing complex impedance values.
:type z: numpy.ndarray(n_frequency, 2, 2) :param z_error: Array containing error values (standard deviation)

of impedance tensor elements.

Parameters

frequency (`np.ndarray(n_frequency)`) – Array of frequency values corresponding to impedance tensor elements.

property det

Determinant of impedance.

property det_error

Return the determinant of impedance error.

property det_model_error

Return the determinant of impedance model error.

estimate_depth_of_investigation()

Estimate depth of investigation. :return: DESCRIPTION. :rtype: TYPE

estimate_dimensionality(skew_threshold=5, eccentricity_threshold=0.1)

Estimate dimensionality of the impedance tensor from parameters such as strike and phase tensor eccentricity :return: DESCRIPTION. :rtype: TYPE

estimate_distortion(n_frequencies=None, comp='det', only_2d=False, clockwise=True)

Estimate distortion. :param only_2d:

Defaults to False.

Parameters

- **n_frequencies** (TYPE, optional) – DESCRIPTION, defaults to None.
- **comp** (TYPE, optional) – DESCRIPTION, defaults to “det”.

Param

DESCRIPTION.

Type

TYPE

Returns
DESCRIPTION.

Return type
TYPE

property invariants

Weaver Invariants.

property phase

Phase of impedance.

property phase_det

Phase determinant.

property phase_error

Phase error of impedance

Uncertainty in phase (in degrees) is computed by defining a circle around the z vector in the complex plane. The uncertainty is the absolute angle between the vector to (x,y) and the vector between the origin and the tangent to the circle..

property phase_error_det

Phase error determinant.

property phase_error_xx

Phase error of xx component.

property phase_error_xy

Phase error of xy component.

property phase_error_yx

Phase error of yx component.

property phase_error_yy

Phase error of yy component.

property phase_model_error

Phase model error of impedance.

property phase_model_error_det

Phase model error determinant.

property phase_model_error_xx

Phase model error of xx component.

property phase_model_error_xy

Phase model error of xy component.

property phase_model_error_yx

Phase model error of yx component.

property phase_model_error_yy

Phase model error of yy component.

property phase_tensor

Phase tensor object based on impedance.

property phase_xx

Phase of xx component.

property phase_xy

Phase of xy component.

property phase_yx

Phase of yx component.

property phase_yy

Phase of yy component.

remove_distortion(*distortion_tensor=None, distortion_error_tensor=None, n_frequencies=None, comp='det', only_2d=False, inplace=False*)

Remove distortion D form an observed impedance tensor Z to obtain the uperturbed “correct” Z0: $Z = D * Z_0$

Propagation of errors/uncertainties included :param only_2d:

Defaults to False.

Parameters

- **comp** – Defaults to “det”.
- **n_frequencies** – Defaults to None.
- **distortion_tensor** (*np.ndarray(2, 2, dtype=real)*, optional) – Real distortion tensor as a 2x2, defaults to None.

:param distortion_error_tensor:, defaults to None. :type distortion_error_tensor: *np.ndarray(2, 2, dtype=real)*, optional :param inplace: Update the current object or return a new impedance, defaults to False. :type inplace: boolean, optional :return s: Input distortion tensor. :rtype s: *np.ndarray(2, 2, dtype='real')* :return s: Impedance tensor with distorion removed. :rtype s: *np.ndarray(num_frequency, 2, 2, dtype='complex')* :return s: Impedance tensor error after distortion is removed. :rtype s: *np.ndarray(num_frequency, 2, 2, dtype='complex')*

remove_ss(*reduce_res_factor_x=1.0, reduce_res_factor_y=1.0, inplace=False*)

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 :

- $Z = S * Z_0$

therefore the correct Z will be :

- $Z_0 = S^{-1} * Z$

Parameters

- **reduce_res_factor_x**(*float or iterable list or array, optional*) – Static shift factor to be applied to x components (ie $z[:, 0, :]$). This is assumed to be in resistivity scale, defaults to 1.0.
- **reduce_res_factor_y**(*float or iterable list or array, optional*) – Static shift factor to be applied to y components (ie $z[:, 1, :]$). This is assumed to be in resistivity scale, defaults to 1.0.
- **inplace** (*boolean, optional*) – Update the current object or return a new impedance, defaults to False.

Return s
Static shift matrix.,,

Rtype s
np.ndarray ((2, 2))

Return s
Corrected Z if inplace is False.

Rtype s
mtpy.core.z.Z

property res_det
Resistivity determinant.

property res_error_det
Resistivity error determinant.

property res_error_xx
Resistivity error of xx component.

property res_error_xy
Resistivity error of xy component.

property res_error_yx
Resistivity error of yx component.

property res_error_yy
Resistivity error of yy component.

property res_model_error_det
Resistivity model error determinant.

property res_model_error_xx
Resistivity model error of xx component.

property res_model_error_xy
Resistivity model error of xy component.

property res_model_error_yx
Resistivity model error of yx component.

property res_model_error_yy
Resistivity model error of yy component.

property res_xx
Resistivity of xx component.

property res_xy
Resistivity of xy component.

property res_yx
Resistivity of yx component.

property res_yy
Resistivity of yy component.

property resistivity
Resistivity of impedance.

property resistivity_error

Resistivity error of impedance

By standard error propagation, relative error in resistivity is 2*relative error in z amplitude..

property resistivity_model_error

Resistivity model error of impedance.

set_resistivity_phase(*resistivity, phase, frequency, res_error=None, phase_error=None, res_model_error=None, phase_model_error=None*)

Set values for resistivity (res - in Ohm m) and phase (phase - in degrees), including error propagation.
:param phase_model_error:

Defaults to None.

Parameters

- **res_model_error** – Defaults to None.
- **resistivity** (*np.ndarray(num_frequency, 2, 2)*) – Resistivity array in Ohm-m.
- **phase** (*np.ndarray(num_frequency, 2, 2)*) – Phase array in degrees.
- **frequency** (*np.ndarray(num_frequency)*) – Frequency array in Hz.
- **res_error** (*np.ndarray(num_frequency, 2, 2), optional*) – Resistivity error array in Ohm-m, defaults to None.
- **phase_error** (*np.ndarray(num_frequency, 2, 2), optional*) – Phase error array in degrees, defaults to None.

property units

impedance units

property z

Impedance tensor

np.ndarray(nfrequency, 2, 2).

property z_error

Error of impedance tensor array as standard deviation.

property z_model_error

Model error of impedance tensor array as standard deviation.

mtpy.gis package

Submodules

mtpy.gis.raster_tools module

Created on Sun May 11 12:15:37 2014

@author: jrpeacock

mtpy.gis.raster_tools.array2raster(*raster_fn, array, lower_left, cell_size_north, cell_size_east, crs, rotation_angle=0*)

Use rasterio to write raster file. :param rotation_angle:

Defaults to 0.

Parameters

- **array**
- **raster_fn** (*TYPE*) – DESCRIPTION.
- **lower_left** (*TYPE*) – DESCRIPTION.
- **cell_size_north** (*TYPE*) – DESCRIPTION.
- **cell_size_east** (*TYPE*) – DESCRIPTION.
- **crs** (*TYPE*) – DESCRIPTION.

Returns

DESCRIPTION.

Return type*TYPE***mtpy.gis.shapefile_creator module****Description:**

Create shape files for Phase Tensor Ellipses, Tipper Real/Imag. export the phase tensor map and tippers into jpeg/png images

CreationDate: 2017-03-06 Developer: fei.zhang@gov.au

Revision History:

LastUpdate: 10/11/2017 FZ fix bugs after the big merge LastUpdate: 20/11/2017 change from freq to period filenames, allow to specify a period LastUpdate: 30/10/2018 combine ellipses and tippers together, refactorings

brenainn.moushall@gov.au 27-03-2020 17:33:23 AEDT:

Fix outfile/directory issue (see commit messages)

update to v2 jpeacock 2024-04-15

class mtpy.gis.shapefile_creator.ShapefileCreator(*mt_dataframe*, *output_crs*, *save_dir=None*)

Bases: object

Create phase tensor and tipper shape files using geopandas and shapely tools.

estimate_arrow_size(*quantile=0.03*)

Arrow size from station distances.

estimate_ellipse_size(*quantile=0.015*)

Estimate ellipse size from station distances.

make_shp_files(*pt=True*, *tipper=True*, *periods=None*, *period_tol=None*)

If you want all stations on the same period map need to interpolate before converting to an MTDataFrame
`md.interpolate(new_periods)` :param period_tol:

Defaults to None.

Parameters

- **periods** – Defaults to None.
- **pt** (*TYPE, optional*) – DESCRIPTION, defaults to True.
- **tipper** (*TYPE, optional*) – DESCRIPTION, defaults to True.

Returns

DESCRIPTION.

Return type

TYPE

property mt_dataframe

MTDataFrame object.

property output_crs

Output crs.

property save_dir

Save dir.

property x_key

X key.

property y_key

Y key.

Module contents

mtpy.imaging package

Subpackages

mtpy.imaging.mtplot_tools package

Submodules

[mtpy.imaging.mtplot_tools.arrows module](#)

Created on Sun Sep 25 15:16:31 2022

@author: jpeacock

class mtpy.imaging.mtplot_tools.arrows.MTArrows(kwargs)**

Bases: `object`

Helper class to read a dictionary of arrow properties.

Arguments::

- ‘size’
[float] multiplier to scale the arrow. *default* is 5
- ‘head_length’
[float] length of the arrow head *default* is 1.5
- ‘head_width’
[float] width of the arrow head *default* is 1.5
- ‘lw’
[float] line width of the arrow *default* is .5
- ‘color’
[tuple (real, imaginary)] color of the arrows for real and imaginary
- ‘threshold’: float
threshold of which any arrow larger than this number will not be plotted, helps clean up if the data is not good. *default* is 1, note this is before scaling by ‘size’

- ‘direction’
[[0 | 1]]
 - 0 for arrows to point away a conductor
 - 1 for arrow to point toward from conductor

mtpy.imaging.mtplot_tools.base module

Base classes for plotting classes

author

jpeacock

class mtpy.imaging.mtplot_tools.base.PlotBase(**kwargs)

Bases: *PlotSettings*

Base class for plotting objects.

plot()

Plot function.

redraw_plot()

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

Example

```
>>> # change the color and marker of the xy components
>>> p1.xy_color = (.5,.5,.9)
>>> p1.xy_marker = '*'
>>> p1.redraw_plot()
```

save_plot(*save_fn*, *file_format*=‘pdf’, *orientation*=‘portrait’, *fig_dpi*=None, *close_plot*=True)

Save_plot will save the figure to *save_fn*.

Arguments:

```
**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_ResPhase.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 fig_dpi will be that at
```

(continues on next page)

(continued from previous page)

which the figure was made. I don't think that it can be larger than fig_dpi of the figure.

```
**close_plot** : [ true | false ]
* True will close the plot after saving.
* False will leave plot open
```

:Example: ::

```
>>> # to save plot as jpg
>>> p1.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

```
>>> [ax.grid(True, which='major') for ax in [p1.axr,p1.axp]]
>>> p1.update_plot()
```

`class mtpy.imaging.mtplot_tools.base.PlotBaseMaps(**kwargs)`

Bases: *PlotBase*

Base object for plot classes that use map views, includes methods for interpolation.

`add_raster(ax, raster_fn, add_colorbar=True, **kwargs)`

Add a raster to an axis using rasterio. :param ax: DESCRIPTION. :type ax: TYPE :param raster_fn: DESCRIPTION. :type raster_fn: TYPE :param add_colorbar: DESCRIPTION, defaults to True. :type add_colorbar: TYPE, optional :param **kwargs: DESCRIPTION. :type **kwargs: TYPE :return: DESCRIPTION. :rtype: TYPE

`static get_interp1d_functions_t(tf, interp_type='slinear')`

Get interp1d functions t. :param tf: :param interp_type: DESCRIPTION, defaults to "slinear". :type interp_type: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

`static get_interp1d_functions_z(tf, interp_type='slinear')`

Get interp1d functions z. :param tf: :param interp_type: DESCRIPTION, defaults to "slinear". :type interp_type: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

`interpolate_to_map(plot_array, component)`

Interpolate points onto a 2d map. :param plot_array: DESCRIPTION. :type plot_array: TYPE :param component: DESCRIPTION. :type component: TYPE :param cell_size: DESCRIPTION002, defaults to 0. :type cell_size: TYPE, optional :param n_padding_cells: DESCRIPTION, defaults to 10. :type n_padding_cells: TYPE, optional :param interpolation_method: DESCRIPTION, defaults to "delaunay". :type interpolation_method: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

`class mtpy.imaging.mtplot_tools.base.PlotBaseProfile(tf_list, **kwargs)`

Bases: *PlotBase*

Base object for profile plots like pseudo sections.

`property rotation_angle`

Rotation angle.

mtpy.imaging.mtplot_tools.ellipses module

Created on Sun Sep 25 15:19:16 2022

@author: jpeacock

```
class mtpy.imaging.mtplot_tools.ellipses.MTEllipse(**kwargs)
```

Bases: object

Helper class for getting ellipse properties from an input dictionary.

Arguments:

```
* 'size' -> size of ellipse in points
    *default* is .25

* 'colorby' : [ 'phimin' | 'phimax' | 'beta' |
    'skew_seg' | 'phidet' | 'ellipticity' ]

    - 'phimin' -> colors by minimum phase
    - 'phimax' -> colors by maximum phase
    - 'skew' -> colors by skew
    - 'skew_seg' -> colors by skew in
        discrete segments
        defined by the range
    - 'normalized_skew' -> colors by
        normalized_skew
        see Booker, 2014
    - 'normalized_skew_seg' -> colors by
        normalized_skew
        discrete segments
        defined by the range
    - 'phidet' -> colors by determinant of
        the phase tensor
    - 'ellipticity' -> colors by ellipticity
    *default* is 'phimin'

* 'range' : tuple (min, max, step)
    Need to input at least the min and max
    and if using 'skew_seg' to plot
    discrete values input step as well
    *default* depends on 'colorby'

* 'cmap' : [ 'mt_yl2rd' | 'mt_b12yl2rd' |
    'mt_wh2bl' | 'mt_rd2bl' |
    'mt_b12wh2rd' | 'mt_seg_b12wh2rd' |
    'mt_rd2gr2bl']

    - 'mt_yl2rd'      --> yellow to red
    - 'mt_b12yl2rd'   --> blue to yellow to red
    - 'mt_wh2bl'       --> white to blue
    - 'mt_rd2bl'       --> red to blue
    - 'mt_b12wh2rd'   --> blue to white to red
    - 'mt_b12gr2rd'   --> blue to green to red
    - 'mt_rd2gr2bl'   --> red to green to blue
```

(continues on next page)

(continued from previous page)

```
- 'mt_seg_bl2wh2rd' --> discrete blue to  
white to red
```

property ellipse_cmap_bounds
Ellipse cmap bounds.

property ellipse_cmap_n_segments
Ellipse cmap n segments.

property ellipse_properties
Ellipse properties.

get_color_map()
Get a color map.

get_pt_color_array(pt_object)
Get the appropriate color by array.

get_range()
Get an appropriate range for the colorby.

mtpy.imaging.mtplot_tools.map_interpolation_tools module

Created on Sun Sep 25 15:06:58 2022

@author: jpeacock

```
mtpy.imaging.mtplot_tools.map_interpolation_tools.get_plot_xy(plot_array, cell_size,  
n_padding_cells)
```

Get plot x and plot y from a plot array to interpolate on to. :param plot_array: DESCRIPTION. :type plot_array: TYPE :param cell_size: DESCRIPTION. :type cell_size: TYPE :param n_padding_cells: DESCRIPTION. :type n_padding_cells: TYPE :return: DESCRIPTION. :rtype: TYPE

```
mtpy.imaging.mtplot_tools.map_interpolation_tools.griddata_interpolate(x, y, values, new_x,  
new_y, interpolation_method='cubic')
```

Griddata interpolate. :param values: :param x: DESCRIPTION. :type x: TYPE :param y: DESCRIPTION. :type y: TYPE :param value: DESCRIPTION. :type value: TYPE :param new_x: DESCRIPTION. :type new_x: TYPE :param new_y: DESCRIPTION. :type new_y: TYPE :param interpolation_method: DESCRIPTION, defaults to “cubic”. :type interpolation_method: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

```
mtpy.imaging.mtplot_tools.map_interpolation_tools.in_hull(p, hull)
```

Test if points in p are within the convex hull.

```
mtpy.imaging.mtplot_tools.map_interpolation_tools.interpolate_to_map(plot_array, component,  
cell_size=0.002,  
n_padding_cells=10,  
interpolation_method='delaunay',  
interpolation_power=5,  
nearest_neighbors=7)
```

Interpolate to map. :param nearest_neighbors:

Defaults to 7.

Parameters

- **interpolation_power** – Defaults to 5.
- **plot_array** (*TYPE*) – DESCRIPTION.
- **component** (*TYPE*) – DESCRIPTION.
- **cell_size** (*TYPE, optional*) – DESCRIPTION002, defaults to 0.002.
- **n_padding_cells** (*TYPE, optional*) – DESCRIPTION, defaults to 10.
- **interpolation_method** (*TYPE, optional*) – DESCRIPTION, defaults to “delaunay”.

Returns

DESCRIPTION.

Return type

TYPE

```
mtpy.imaging.mtplot_tools.map_interpolation_tools.interpolate_to_map_griddata(plot_array,
                                         component,
                                         cell_size=0.002,
                                         n_padding_cells=10,
                                         interpolation_method='cubic')
```

Interpolate using scipy.interpolate.griddata. :param interpolation_method:

Defaults to “cubic”.

Parameters

- **plot_array** (*TYPE*) – DESCRIPTION.
- **component** (*TYPE*) – DESCRIPTION.
- **cell_size** (*TYPE, optional*) – DESCRIPTION002, defaults to 0.002.
- **n_padding_cells** (*TYPE, optional*) – DESCRIPTION, defaults to 10.

Returns

DESCRIPTION.

Return type

TYPE

```
mtpy.imaging.mtplot_tools.map_interpolation_tools.interpolate_to_map_triangulate(plot_array,
                                         component,
                                         cell_size=0.002,
                                         n_padding_cells=10,
                                         nearest_neighbors=7,
                                         interp_pow=4)
```

plot_array must have key words:

- **latitude**: latitude in decimal degrees of measured points
- **longitude**: longitude in decimal degrees of measured points
- values should have the proper name of the input component. For example if you are plotting the resistivity of the xy component then the keyword should be ‘res_xy’.

Parameters

- **plot_x** (*np.ndarray*) – Desired regular x locations to interpolate to.
- **plot_y** (*np.ndarray*) – Desired regular x locations to interpolate to.
- **plot_array** (*np.ndarray*) – Structured array, see above.
- **component** (*string*) – Component or keyword of the plot_array to plot.
- **cell_size** (*TYPE, optional*) – Size of cells 002, defaults to 0.002.
- **n_padding_cells** (*TYPE, optional*) – DESCRIPTION, defaults to 10.
- **nearest_neighbors** (*TYPE, optional*) – DESCRIPTION, defaults to 7.
- **interp_pow** (*TYPE, optional*) – DESCRIPTION, defaults to 4.

Returns

DESCRIPTION.

Return type

TYPE

```
mtpy.imaging.mtplot_tools.map_interpolation_tools.triangulate_interpolation(x, y, values,  
                           padded_x,  
                           padded_y,  
                           new_x, new_y,  
                           near-  
                           est_neighbors=7,  
                           interp_pow=4)
```

Triangulate interpolation. :param interp_pow:

Defaults to 4.

Parameters

- **nearest_neighbors** – Defaults to 7.
- **padded_y**
- **padded_x**
- **x** (*TYPE*) – DESCRIPTION.
- **y** (*TYPE*) – DESCRIPTION.
- **values** (*TYPE*) – DESCRIPTION.
- **new_x** (*TYPE*) – DESCRIPTION.
- **new_y** (*TYPE*) – DESCRIPTION.

Param

DESCRIPTION.

Type

TYPE

Returns

DESCRIPTION.

Return type

TYPE

mtpy.imaging.mtplot_tools.plot_settings module

Created on Sun Sep 25 15:20:43 2022

@author: jpeacock

class mtpy.imaging.mtplot_tools.plot_settings.PlotSettings(kwargs)**

Bases: *MTArrows*, *MTEllipse*

Hold all the plot settings that one might need.

property arrow_imag_properties

Arrow imag properties.

property arrow_real_properties

Arrow real properties.

property det_error_bar_properties

Xy error bar properties for xy mode. :return: DESCRIPTION. :rtype: TYPE

property font_dict

make_pt_cb(ax)

Make pt cb.

property period_label_dict

Log 10 labels. :return: DESCRIPTION. :rtype: TYPE

set_period_limits(period)

Set period limits. :return: DESCRIPTION. :rtype: TYPE

set_phase_limits(phase, mode='od')

Set phase limits.

set_resistivity_limits(resistivity, mode='od', scale='log')

Set resistivity limits. :param scale:

Defaults to “log”.

Parameters

- **resistivity** (TYPE) – DESCRIPTION.
- **mode** (TYPE, optional) – DESCRIPTION, defaults to “od”.

Returns

DESCRIPTION.

Return type

TYPE

property text_dict

Text dict.

property xy_error_bar_properties

Xy error bar properties for xy mode. :return: DESCRIPTION. :rtype: TYPE

property yx_error_bar_properties

Xy error bar properties for xy mode. :return: DESCRIPTION. :rtype: TYPE

mtpy.imaging.mtplot_tools.plotters module

Simple plotters elements that can be assembled in various plotting classes

Created on Sun Sep 25 15:27:28 2022

author

jpeacock

`mtpy.imaging.mtplot_tools.plotters.add_raster(ax, raster_fn, add_colorbar=True, **kwargs)`

Add a raster to an axis using rasterio

Parameters

- **raster_fn** (*TYPE*) – DESCRIPTION
- ****kwargs** – DESCRIPTION

Returns

DESCRIPTION

Return type

TYPE

`mtpy.imaging.mtplot_tools.plotters.plot_errorbar(ax, x_array, y_array, y_error=None, x_error=None, **kwargs)`

convinience function to make an error bar instance

Arguments:

ax

[matplotlib.axes instance] axes to put error bar plot on

x_array

[np.ndarray(nx)] array of x values to plot

y_array

[np.ndarray(nx)] array of y values to plot

y_error

[np.ndarray(nx)] array of errors in y-direction to plot

x_error

[np.ndarray(ns)] array of error in x-direction to plot

color

[string or (r, g, b)] color of marker, line and error bar

marker

[string] marker type to plot data as

mew

[string] marker edgewidth

ms

[float] size of marker

ls

[string] line style between markers

lw

[float] width of line between markers

e_capsize
[float] size of error bar cap

e_capthick
[float] thickness of error bar cap

picker
[float] radius in points to be able to pick a point.

Returns:

errorbar_object
[matplotlib.Axes.errorbar] error bar object containing line data, errorbars, etc.

`mtpy.imaging.mtplot_tools.plotters.plot_phase(ax, period, phase, error, yx=False, **properties)`

plot apparent resistivity to the given axis with given properties

Parameters

- **ax** (TYPE) – DESCRIPTION
- **resistivity** (TYPE) – DESCRIPTION
- **period** (TYPE) – DESCRIPTION
- ****properties** – DESCRIPTION

Returns

DESCRIPTION

Return type

TYPE

`mtpy.imaging.mtplot_tools.plotters.plot_pt_lateral(ax, pt_obj, color_array, ellipse_properties, y_shift=0, fig=None, edge_color=None, n_index=0)`

Parameters

- **ax** (TYPE) – DESCRIPTION
- **pt_obj** (TYPE) – DESCRIPTION
- **color_array** (TYPE) – DESCRIPTION
- **ellipse_properties** (TYPE) – DESCRIPTION
- **bounds** (TYPE, optional) – DESCRIPTION, defaults to None

Returns

DESCRIPTION

Return type

TYPE

`mtpy.imaging.mtplot_tools.plotters.plot_resistivity(ax, period, resistivity, error, **properties)`

plot apparent resistivity to the given axis with given properties

Parameters

- **ax** (TYPE) – DESCRIPTION
- **resistivity** (TYPE) – DESCRIPTION
- **period** (TYPE) – DESCRIPTION

- ****properties** – DESCRIPTION

Returns

DESCRIPTION

Return type

TYPE

```
mtpy.imaging.mtplot_tools.plotters.plot_tipper_lateral(axt, t_obj, plot_tipper, real_properties,  
                                                       imag_properties, font_size=6, legend=True,  
                                                       zero_reference=False, arrow_direction=1)
```

Parameters

- **axt** (TYPE) – DESCRIPTION
- **t_obj** (TYPE) – DESCRIPTION

Returns

DESCRIPTION

Return type

TYPE

mtpy.imaging.mtplot_tools.utils module

Utility functions for plotting

Created on Sun Sep 25 15:49:01 2022

author

jpeacock

```
mtpy.imaging.mtplot_tools.utils.add_colorbar_axis(ax, fig)
```

```
mtpy.imaging.mtplot_tools.utils.get_log_tick_labels(ax, spacing=1)
```

Parameters

ax (TYPE) – DESCRIPTION

Returns

DESCRIPTION

Return type

TYPE

```
mtpy.imaging.mtplot_tools.utils.get_period_limits(period)
```

```
mtpy.imaging.mtplot_tools.utils.make_color_list(cbax, nseg, ckmin, ckmax, ckstep)
```

```
mtpy.imaging.mtplot_tools.utils.make_value_str(value, value_list=None, spacing='{0:^8}',  
                                               value_format='{0: .2f}', append=False, add=False)
```

helper function for writing values to a file, takes in a value and either appends or adds value to value_list according to the spacing and format of the string.

Arguments:

value : float

value_list : list of values converted to strings

spacing

[spacing of the string that the value will be converted] to.

value_format

[format of the string that the value is being] converted to.

append

[[True | False]] if True then appends the value to value list

add

[[True | False]] if True adds value string to the other value strings in value_list

Returns:**value_list**

[the input value_list with the new value either] added or appended.

or

value_str : value string if add and append are false

```
mtpy.imaging.mtplot_tools.utils.round_to_step(num, base=5)
```

Module contents

```
class mtpy.imaging.mtplot_tools.MTArrows(**kwargs)
```

Bases: object

Helper class to read a dictionary of arrow properties.

Arguments::

- ‘size’
[float] multiplier to scale the arrow. *default* is 5
- ‘head_length’
[float] length of the arrow head *default* is 1.5
- ‘head_width’
[float] width of the arrow head *default* is 1.5
- ‘lw’
[float] line width of the arrow *default* is .5
- ‘color’
[tuple (real, imaginary)] color of the arrows for real and imaginary
- ‘threshold’: float
threshold of which any arrow larger than this number will not be plotted, helps clean up if the data is not good. *default* is 1, note this is before scaling by ‘size’
- ‘direction’
[[0 | 1]]
 - 0 for arrows to point away a conductor
 - 1 for arrow to point toward from conductor

```
class mtpy.imaging.mtplot_tools.MTEllipse(**kwargs)
```

Bases: object

Helper class for getting ellipse properties from an input dictionary.

Arguments:

```
* 'size' -> size of ellipse in points
    *default* is .25

* 'colorby' : [ 'phimin' | 'phimax' | 'beta' |
    'skew_seg' | 'phidet' | 'ellipticity' ]

    - 'phimin' -> colors by minimum phase
    - 'phimax' -> colors by maximum phase
    - 'skew' -> colors by skew
    - 'skew_seg' -> colors by skew in
        discrete segments
        defined by the range
    - 'normalized_skew' -> colors by
        normalized_skew
        see Booker, 2014
    - 'normalized_skew_seg' -> colors by
        normalized_skew
        discrete segments
        defined by the range
    - 'phidet' -> colors by determinant of
        the phase tensor
    - 'ellipticity' -> colors by ellipticity
    *default* is 'phimin'

* 'range' : tuple (min, max, step)
    Need to input at least the min and max
    and if using 'skew_seg' to plot
    discrete values input step as well
    *default* depends on 'colorby'

* 'cmap' : [ 'mt_yl2rd' | 'mt_b12yl2rd' |
    'mt_wh2bl' | 'mt_rd2bl' |
    'mt_b12wh2rd' | 'mt_seg_b12wh2rd' |
    'mt_rd2gr2bl']

    - 'mt_yl2rd'      --> yellow to red
    - 'mt_b12yl2rd'   --> blue to yellow to red
    - 'mt_wh2bl'       --> white to blue
    - 'mt_rd2bl'       --> red to blue
    - 'mt_b12wh2rd'   --> blue to white to red
    - 'mt_b12gr2rd'   --> blue to green to red
    - 'mt_rd2gr2bl'   --> red to green to blue
    - 'mt_seg_b12wh2rd' --> discrete blue to
        white to red
```

property ellipse_cmap_bounds

Ellipse cmap bounds.

property ellipse_cmap_n_segments

Ellipse cmap n segments.

property ellipse_properties

Ellipse properties.

```
get_color_map()
    Get a color map.

get_pt_color_array(pt_object)
    Get the appropriate color by array.

get_range()
    Get an appropriate range for the colorby.

class mtpy.imaging.mtplot_tools.PlotBase(**kwargs)
    Bases: PlotSettings
    Base class for plotting objects.

plot()
    Plot function.

redraw_plot()
    Use this function if you updated some attributes and want to re-plot.
```

Example

```
>>> # change the color and marker of the xy components
>>> p1.xy_color = (.5,.5,.9)
>>> p1.xy_marker = '*'
>>> p1.redraw_plot()
```

save_plot(save_fn, file_format='pdf', orientation='portrait', fig_dpi=None, close_plot=True)

Save_plot will save the figure to save_fn.

Arguments:

```
**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_ResPhase.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 fig_dpi will be that at
    which the figure was made. I don't think that
    it can be larger than fig_dpi of the figure.

**close_plot** : [ true | false ]
```

(continues on next page)

(continued from previous page)

* **True** will close the plot after saving.
 * **False** will leave plot **open**

:Example::

```
>>> # to save plot as jpg
>>> p1.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

```
>>> [ax.grid(True, which='major') for ax in [p1.axr,p1.axp]]
>>> p1.update_plot()
```

class mtpy.imaging.mtplot_tools.PlotBaseMaps(kwargs)**

Bases: *PlotBase*

Base object for plot classes that use map views, includes methods for interpolation.

add_raster(ax, raster_fn, add_colorbar=True, **kwargs)

Add a raster to an axis using rasterio. :param ax: DESCRIPTION. :type ax: TYPE :param raster_fn: DESCRIPTION. :type raster_fn: TYPE :param add_colorbar: DESCRIPTION, defaults to True. :type add_colorbar: TYPE, optional :param **kwargs: DESCRIPTION. :type **kwargs: TYPE :return: DESCRIPTION. :rtype: TYPE

static get_interp1d_functions_t(tf, interp_type='slinear')

Get interp1d functions t. :param tf: :param interp_type: DESCRIPTION, defaults to “slinear”. :type interp_type: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

static get_interp1d_functions_z(tf, interp_type='slinear')

Get interp1d functions z. :param tf: :param interp_type: DESCRIPTION, defaults to “slinear”. :type interp_type: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

interpolate_to_map(plot_array, component)

Interpolate points onto a 2d map. :param plot_array: DESCRIPTION. :type plot_array: TYPE :param component: DESCRIPTION. :type component: TYPE :param cell_size: DESCRIPTION002, defaults to 0. :type cell_size: TYPE, optional :param n_padding_cells: DESCRIPTION, defaults to 10. :type n_padding_cells: TYPE, optional :param interpolation_method: DESCRIPTION, defaults to “delaunay”. :type interpolation_method: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

class mtpy.imaging.mtplot_tools.PlotBaseProfile(tf_list, **kwargs)

Bases: *PlotBase*

Base object for profile plots like pseudo sections.

property rotation_angle

Rotation angle.

class mtpy.imaging.mtplot_tools.PlotSettings(kwargs)**

Bases: *MTArrows, MTEllipse*

Hold all the plot settings that one might need.

```
property arrow_imag_properties
    Arrow imag properties.

property arrow_real_properties
    Arrow real properties.

property det_error_bar_properties
    Xy error bar properties for xy mode. :return: DESCRIPTION. :rtype: TYPE

property font_dict

make_pt_cb(ax)
    Make pt cb.

property period_label_dict
    Log 10 labels. :return: DESCRIPTION. :rtype: TYPE

set_period_limits(period)
    Set period limits. :return: DESCRIPTION. :rtype: TYPE

set_phase_limits(phase, mode='od')
    Set phase limits.

set_resistivity_limits(resistivity, mode='od', scale='log')
    Set resistivity limits. :param scale:
        Defaults to "log".
```

Parameters

- **resistivity** (TYPE) – DESCRIPTION.
- **mode** (TYPE, optional) – DESCRIPTION, defaults to “od”.

Returns

DESCRIPTION.

Return type

TYPE

```
property text_dict
```

Text dict.

```
property xy_error_bar_properties
```

Xy error bar properties for xy mode. :return: DESCRIPTION. :rtype: TYPE

```
property yx_error_bar_properties
```

Xy error bar properties for xy mode. :return: DESCRIPTION. :rtype: TYPE

```
mtpy.imaging.mtplot_tools.add_raster(ax, raster_fn, add_colorbar=True, **kwargs)
```

Add a raster to an axis using rasterio

Parameters

- **raster_fn** (TYPE) – DESCRIPTION
- ****kwargs** – DESCRIPTION

Returns

DESCRIPTION

Return type

TYPE

```
mtpy.imaging.mplot_tools.get_log_tick_labels(ax, spacing=1)
```

Parameters

ax (TYPE) – DESCRIPTION

Returns

DESCRIPTION

Return type

TYPE

```
mtpy.imaging.mplot_tools.griddata_interpolate(x, y, values, new_x, new_y,  
interpolation_method='cubic')
```

Griddata interpolate. :param values: :param x: DESCRIPTION. :type x: TYPE :param y: DESCRIPTION. :type y: TYPE :param value: DESCRIPTION. :type value: TYPE :param new_x: DESCRIPTION. :type new_x: TYPE :param new_y: DESCRIPTION. :type new_y: TYPE :param interpolation_method: DESCRIPTION, defaults to “cubic”. :type interpolation_method: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

```
mtpy.imaging.mplot_tools.interpolate_to_map(plot_array, component, cell_size=0.002,  
n_padding_cells=10, interpolation_method='delaunay',  
interpolation_power=5, nearest_neighbors=7)
```

Interpolate to map. :param nearest_neighbors:

Defaults to 7.

Parameters

- **interpolation_power** – Defaults to 5.
- **plot_array** (TYPE) – DESCRIPTION.
- **component** (TYPE) – DESCRIPTION.
- **cell_size** (TYPE, optional) – DESCRIPTION002, defaults to 0.002.
- **n_padding_cells** (TYPE, optional) – DESCRIPTION, defaults to 10.
- **interpolation_method** (TYPE, optional) – DESCRIPTION, defaults to “delaunay”.

Returns

DESCRIPTION.

Return type

TYPE

```
mtpy.imaging.mplot_tools.plot_errorbar(ax, x_array, y_array, y_error=None, x_error=None, **kwargs)
```

convinience function to make an error bar instance

Arguments:

ax

[matplotlib.axes instance] axes to put error bar plot on

x_array

[np.ndarray(nx)] array of x values to plot

y_array

[np.ndarray(nx)] array of y values to plot

y_error
[np.ndarray(nx)] array of errors in y-direction to plot

x_error
[np.ndarray(ns)] array of error in x-direction to plot

color
[string or (r, g, b)] color of marker, line and error bar

marker
[string] marker type to plot data as

mew
[string] marker edgewith

ms
[float] size of marker

ls
[string] line style between markers

lw
[float] width of line between markers

e_capsize
[float] size of error bar cap

e_capthick
[float] thickness of error bar cap

picker
[float] radius in points to be able to pick a point.

Returns:

errorbar_object
[matplotlib.Axes.errorbar] error bar object containing line data, errorbars, etc.

`mtpy.imaging.mtplot_tools.plot_phase(ax, period, phase, error, yx=False, **properties)`
plot apparent resistivity to the given axis with given properties

Parameters

- **ax (TYPE)** – DESCRIPTION
- **resistivity (TYPE)** – DESCRIPTION
- **period (TYPE)** – DESCRIPTION
- ****properties** – DESCRIPTION

Returns

DESCRIPTION

Return type

TYPE

`mtpy.imaging.mtplot_tools.plot_pt_lateral(ax, pt_obj, color_array, ellipse_properties, y_shift=0, fig=None, edge_color=None, n_index=0)`

Parameters

- **ax (TYPE)** – DESCRIPTION

- **pt_obj** (*TYPE*) – DESCRIPTION
- **color_array** (*TYPE*) – DESCRIPTION
- **ellipse_properties** (*TYPE*) – DESCRIPTION
- **bounds** (*TYPE, optional*) – DESCRIPTION, defaults to None

Returns

DESCRIPTION

Return type

TYPE

`mtpy.imaging.mtplot_tools.plot_resistivity(ax, period, resistivity, error, **properties)`

plot apparent resistivity to the given axis with given properties

Parameters

- **ax** (*TYPE*) – DESCRIPTION
- **resistivity** (*TYPE*) – DESCRIPTION
- **period** (*TYPE*) – DESCRIPTION
- ****properties** – DESCRIPTION

Returns

DESCRIPTION

Return type

TYPE

`mtpy.imaging.mtplot_tools.plot_tipper_lateral(axt, t_obj, plot_tipper, real_properties, imag_properties, font_size=6, legend=True, zero_reference=False, arrow_direction=1)`

Parameters

- **axt** (*TYPE*) – DESCRIPTION
- **t_obj** (*TYPE*) – DESCRIPTION

Returns

DESCRIPTION

Return type

TYPE

`mtpy.imaging.mtplot_tools.triangulate_interpolation(x, y, values, padded_x, padded_y, new_x, new_y, nearest_neighbors=7, interp_pow=4)`

Triangulate interpolation. :param interp_pow:

Defaults to 4.

Parameters

- **nearest_neighbors** – Defaults to 7.
- **padded_y**
- **padded_x**
- **x** (*TYPE*) – DESCRIPTION.
- **y** (*TYPE*) – DESCRIPTION.

- **values** (TYPE) – DESCRIPTION.
- **new_x** (TYPE) – DESCRIPTION.
- **new_y** (TYPE) – DESCRIPTION.

Param

DESCRIPTION.

Type

TYPE

Returns

DESCRIPTION.

Return type

TYPE

Submodules**mtpy.imaging.mtcOLORS module**

Created on Tue May 14 18:05:59 2013

@author: jpeacock-pr

```
class mtpy.imaging.mtcOLORS.FixPointNormalize(vmin=None, vmax=None, sealevel=0, col_val=0.21875,
                                                clip=False)
```

Bases: Normalize

Inspired by <https://stackoverflow.com/questions/20144529/shifted-colorbar-matplotlib> Subclassing Normalize to obtain a colormap with a fixpoint somewhere in the middle of the colormap. This may be useful for a *terrain* map, to set the “sea level” to a color in the blue/turquoise range.

```
mtpy.imaging.mtcOLORS.cmap_discretize(cmap, N)
```

Return a discrete colormap from the continuous colormap cmap.

cmap: colormap instance, eg. cm.jet. N: number of colors.

Example

```
x = resize(arange(100), (5,100)) djet = cmap_discretize(cm.jet, 5) imshow(x, cmap=djet)
```

```
mtpy.imaging.mtcOLORS.get_color(cvar, cmap)
```

Gets the color to plot for the given color map.

```
mtpy.imaging.mtcOLORS.get_plot_color(colorx, comp, cmap, ckmin=None, ckmax=None, bounds=None)
```

Gets the color for the given component, color array and cmap

Note: we now use the linearSegmentedColorMap objects, instead of the get_color function.

```
mtpy.imaging.mtcOLORS.register_cmaps(cmap_dict)
```

Register cmaps.

mtpy.imaging.plot_mt_response module**plot_mt_response**

Plots the resistivity and phase for different modes and components

Created 2017

@author: jpeacock

```
class mtpy.imaging.plot_mt_response.PlotMTResponse(z_object=None, t_object=None, pt_obj=None, station='MT Response', **kwargs)
```

Bases: *PlotBase*

Plots Resistivity and phase for the different modes of the MT response.

At

the moment it supports the input of an .edi file. Other formats that will

be supported are the impedance tensor and errors with an array of periods and .j format.

The normal use is to input an .edi file, however it would seem that not everyone uses this format, so you can input the data and put it into arrays or objects like class mtpy.core.z.Z. Or if the data is in resistivity and phase format they can be input as arrays or a class mtpy.imaging.mtplot.ResPhase. Or you can put it into a class mtpy.imaging.mtplot.MTplot.

The plot places the apparent resistivity in log scale in the top panel(s), depending on the plot_num. The phase is below this, note that 180 degrees has been added to the yx phase so the xy and yx phases plot in the same quadrant. Both the resistivity and phase share the same x-axis which is in log period, short periods on the left to long periods on the right. So if you zoom in on the plot both plots will zoom in to the same x-coordinates. If there is tipper information, you can plot the tipper as a third panel at the bottom, and also shares the x-axis. The arrows are in the convention of pointing towards a conductor. The xx and yy components can be plotted as well, this adds two panels on the right. Here the phase is left unwrapped. Other parameters can be added as subplots such as strike, skew and phase tensor ellipses.

To manipulate the plot you can change any of the attributes listed below and call redraw_plot(). If you know more about matplotlib and want to change axes parameters, that can be done by changing the parameters in the axes attributes and then call update_plot(), note the plot must be open.

property period

Plot period.

plot()

PlotResPhase(filename,fig_num) will plot the apparent resistivity and phase for a single station.

property plot_model_error

Plot model error.

property rotation_angle

Rotation angle.

mtpy.imaging.plot_mt_responses module

plots multiple MT responses simultaneously

Created on Thu May 30 17:02:39 2013 @author: jpeacock-pr

YG: the code there is massey, todo may need to rewrite it sometime

```
class mtpy.imaging.plot_mt_responses.PlotMultipleResponses(mt_data, **kwargs)
```

Bases: *PlotBase*

Plots multiple MT responses simultaneously either in single plots or in one plot of sub-figures or in a single plot with subfigures for each component.

Arguments::

fn_list

[list of filenames to plot] ie. [fn_1, fn_2, ...], *default* is None

plot_num

[[1 | 2 | 3]]

- 1 for just Ex/By and Ey/Bx *default*
- 2 for all 4 components
- 3 for off diagonal plus the determinant

plot_style

[[‘1’ | ‘all’ | ‘compare’]]

determines the plotting style:

- ‘1’ **for plotting each station in a different figure.** *default*
- ‘all’ **for plotting each station in a subplot** all in the same figure
- ‘compare’ **for comparing the responses all in** one plot. Here the responses are colored from dark to light. This plot can get messy if too many stations are plotted.

plot()

Plot the apparent resistivity and phase.

property plot_model_error

Plot model error.

property rotation_angle

Rotation angle.

mtpy.imaging.plot_penetration_depth_1d module

Created on Thu Sep 22 10:58:58 2022

@author: jpeacock

class mtpy.imaging.plot_penetration_depth_1d.PlotPenetrationDepth1D(*tf*, *kwargs*)**

Bases: *PlotBase*

Plot the depth of penetration based on the Niblett-Bostick approximation.

property depth_units

Depth units.

plot()

Plot the depth of investigation as a 1d plot with period on the y-axis and depth on the x axis :return: DESCRIPTION. :rtype: TYPE

mtpy.imaging.plot_penetration_depth_map module

Created on Thu Sep 22 10:58:58 2022

@author: jpeacock

```
class mtpy.imaging.plot_penetration_depth_map.PlotPenetrationDepthMap(mt_data, **kwargs)
Bases: PlotBaseMaps

Plot the depth of penetration based on the Niblett-Bostick approximation.

property depth_units
    Depth units.

plot()
    Plot the depth of investigation as a 1d plot with period on the y-axis and depth on the x axis :return: DESCRIPTION. :rtype: TYPE
```

mtpy.imaging.plot_phase_tensor_maps module

Plot phase tensor map in Lat-Lon Coordinate System

Revision History:

Created by @author: jpeacock-pr on Thu May 30 18:20:04 2013

Modified by Fei.Zhang@gov.au 2017-03:

brenainn.moushall 26-03-2020 15:07:14 AEDT:

Add plotting of geotiff as basemap background.

Updated 2022 by J. Peacock to work with v2

- Using rasterio to plot geotiffs
- factorized
- using interp function for faster plotting.

```
class mtpy.imaging.plot_phase_tensor_maps.PlotPhaseTensorMaps(mt_data, **kwargs)
```

Bases: PlotBaseMaps

Plots phase tensor ellipses in map view from a list of edi files.

property map_scale

Map scale.

```
plot(fig=None, save_path=None, show=True, raster_file=None, raster_kwarg={})
```

Plots the phase tensor map. :param raster_kwarg:

Defaults to {}.

Parameters

- **raster_file** – Defaults to None.
- **fig** – Optional figure object, defaults to None.
- **save_path** – Path to folder for saving plots, defaults to None.
- **show** – Show plots if True, defaults to True.
- **raster_dict** – Dictionary containing information for a raster to be plotted below the phase tensors.

property rotation_angle

Rotation angle.

property skew_cmap_bounds

Skew bounds for a segmented colorbar.

mtpy.imaging.plot_phase_tensor_pseudosection module

Created on Thu May 30 18:10:55 2013

@author: jpeacock-pr

```
class mtpy.imaging.plot_phase_tensor_pseudosection.PlotPhaseTensorPseudoSection(mt_data,
**kwargs)
```

Bases: *PlotBaseProfile*

PlotPhaseTensorPseudoSection will plot the phase tensor ellipses in a pseudo section format

To get a list of .edi files that you want to plot -> :Example:

```
>>> import mtpy.imaging.mtplot as mtplot
>>> import os
>>> edipath = r"/home/EDIfiles"
>>> edilist = [os.path.join(edipath,edi) for edi in os.listdir(edipath)
>>> ...           if edi.find('.edi')>0]
```

- If you want to plot minimum phase colored from blue to red in a range of

20 to 70 degrees you can do it one of two ways->

1) :Example:

```
>>> edict = {'range':(20,70), 'cmap':'mt_b12gr2rd','colorby':'phimin'}
>>> pt1 = mtplot.plot_pt_pseudosection(fn_list=edilist,
                                         ellipse_dict=edict)
```

2) :Example:

```
>>> pt1 = mtplot.plot_pt_pseudosection(fn_list=edilist, plot_yn='n')
>>> pt1.ellipse_colorby = 'phimin'
>>> pt1.ellipse_cmap = 'mt_b12gr2rd'
>>> pt1.ellipse_range = (20,70)
>>> pt1.plot()
```

- If you want to add real induction arrows that are scaled by 10 and point

away from a conductor ->

Example

```
>>> pt1.plot_tipper = 'yr'
>>> pt1.arrow_size = 10
>>> pt1.arrow_direction = -1
>>> pt1.redraw_plot()
```

- If you want to save the plot as a pdf with a generic name ->

Example

```
::    >>> pt1.save_figure(r"/home/PTFigures", file_format='pdf', dpi=300) File saved to
'./home/PTFigures/PTPseudoSection.pdf'
```

plot()

Plots the phase tensor pseudo section.

See class doc string for
more details.

mtpy.imaging.plot_pseudosection module

Created on Thu May 30 18:39:58 2013

@author: jpeacock-pr

class mtpy.imaging.plot_pseudosection.PlotResPhasePseudoSection(mt_data, **kwargs)

Bases: *PlotBaseProfile*

Plot a resistivity and phase pseudo section for different components

Need to input one of the following lists::

plot()

Plot function. :return: DESCRIPTION. :rtype: TYPE

mtpy.imaging.plot_pt module

Created on Thu May 30 17:07:50 2013

@author: jpeacock

class mtpy.imaging.plot_pt.PlotPhaseTensor(pt_object, station=None, **kwargs)

Bases: *PlotBase*

Will plot phase tensor, strike angle, min and max phase angle, azimuth, skew, and ellipticity as subplots on one plot. It can plot the resistivity tensor along side the phase tensor for comparison.

plot(rotation_angle=None)

Plots the phase tensor elements.

property rotation_angle

Rotation angle.

mtpy.imaging.plot_residual_pt_maps module

PlotResidualPhaseTensor

*plots the residual phase tensor for two different sets of measurements

Created on Wed Oct 16 14:56:04 2013

@author: jpeacock-pr

```
class mtpy.imaging.plot_residual_pt_maps.PlotResidualPTMaps(mt_data_01, mt_data_02,
                                                               frequencies=array([1.00000000e-03,
                                                                     1.42510267e-03, 2.03091762e-03,
                                                                     2.89426612e-03, 4.12462638e-03,
                                                                     5.87801607e-03, 8.37677640e-03,
                                                                     1.19377664e-02, 1.70125428e-02,
                                                                     2.42446202e-02, 3.45510729e-02,
                                                                     4.92388263e-02, 7.01703829e-02,
                                                                     1.00000000e-01, 1.42510267e-01,
                                                                     2.03091762e-01, 2.89426612e-01,
                                                                     4.12462638e-01, 5.87801607e-01,
                                                                     8.37677640e-01, 1.19377664e+00,
                                                                     1.70125428e+00, 2.42446202e+00,
                                                                     3.45510729e+00, 4.92388263e+00,
                                                                     7.01703829e+00, 1.00000000e+01,
                                                                     1.42510267e+01, 2.03091762e+01,
                                                                     2.89426612e+01, 4.12462638e+01,
                                                                     5.87801607e+01, 8.37677640e+01,
                                                                     1.19377664e+02, 1.70125428e+02,
                                                                     2.42446202e+02, 3.45510729e+02,
                                                                     4.92388263e+02, 7.01703829e+02,
                                                                     1.00000000e+03]), **kwargs)
```

Bases: *PlotBase*

This will plot residual phase tensors in a map for a single frequency.

The data is read in and stored in 2 ways, one as a list ResidualPhaseTensor object for each matching station and the other in a structured array with all the important information. The structured array is the one that is used for plotting. It is computed each time plot() is called so if it is manipulated it is reset. The array is sorted by relative offset, so no special order of input is needed for the file names. However, the station names should be verbatim between surveys, otherwise it will not work.

The residual phase tensor is calculated as $I - (\Phi_2)^{-1}(\Phi_1)$

The default coloring is by the geometric mean as $\sqrt{\Phi_{\min} \cdot \Phi_{\max}}$, which defines the percent change between measurements.

There are a lot of parameters to change how the plot looks, have a look below if you figure looks a little funny. The most useful will be ellipse_size

The ellipses are normalized by the largest Phi_max of the survey.

Example

```
>>> import mtpy.imaging.mtplot as mtplot
>>> import os
>>> edipath1 = r"/home/EDIfiles1"
>>> edilist1 = [os.path.join(edipath1, edi) for edi in os.
   >>>     .listdir(edipath1)
>>> ...      if edi.find('.edi')>0]
>>> edipath2 = r"/home/EDIfiles2"
>>> edilist2 = [os.path.join(edipath2, edi) for edi in os.
   >>>     .listdir(edipath2)
>>> ...      if edi.find('.edi')>0]
>>> # color by phimin with a range of 0-5 deg
>>> ptmap = mtplot.plot_residual_pt_maps(edilist1, edilist2,_
   >>> freqspot=10,
```

(continues on next page)

(continued from previous page)

```
>>> ...
>>> ...
>>>
>>> #
>>> #---add an image---
>>> ptmap.image_file = r"/home/Maps/Basemap.jpg"
>>> ptmap.image_extent = (0,10,0,10)
>>> ptmap.redraw_plot()
>>> #
>>> #---Save the plot---
>>> ptmap.save_plot(r"/home/EDIfiles",file_format='pdf')
>>> 'Saved figure to /home/EDIfile/PTMaps/PTmap_phimin_10.0_Hz.pdf'
```

Example

```
>>> #change the axis label and grid color
>>> ptmap.ax.set_xlabel('Latitude (deg)')
>>> ptmap.ax.grid(which='major', color=(.5,1,0))
>>> ptmap.update_plot()
```

Example

```
>>> # plot seismic hypocenters from a file
>>> lat, lon, depth = np.loadtxt(r"/home/seismic_hypocenter.txt")
>>> ptmap.ax.scatter(lon, lat, marker='o')
```

property map_scale

Map scale.

plot()

Plot residual phase tensor.

property rotation_angle

Rotation angle.

mtpy.imaging.plot_residual_pt_ps module

PlotResidualPhaseTensorPseudoSection

*plots the residual phase tensor for two different sets of measurements

Created on Wed Oct 16 14:56:04 2013

@author: jpeacock-pr

```
class mtpy.imaging.plot_residual_pt_ps.PlotResidualPTPseudoSection(mt_data_01, mt_data_02,
    frequencies=array([1.0000000e-03, 1.42510267e-03, 2.03091762e-03, 2.89426612e-03, 4.12462638e-03, 5.87801607e-03, 8.37677640e-03, 1.19377664e-02, 1.70125428e-02, 2.42446202e-02, 3.45510729e-02, 4.92388263e-02, 7.01703829e-02, 1.00000000e-01, 1.42510267e-01, 2.03091762e-01, 2.89426612e-01, 4.12462638e-01, 5.87801607e-01, 8.37677640e-01, 1.19377664e+00, 1.70125428e+00, 2.42446202e+00, 3.45510729e+00, 4.92388263e+00, 7.01703829e+00, 1.00000000e+01, 1.42510267e+01, 2.03091762e+01, 2.89426612e+01, 4.12462638e+01, 5.87801607e+01, 8.37677640e+01, 1.19377664e+02, 1.70125428e+02, 2.42446202e+02, 3.45510729e+02, 4.92388263e+02, 7.01703829e+02, 1.00000000e+03]), **kwargs)
```

Bases: *PlotBaseProfile*

This will plot residual phase tensors in a pseudo section.

The data is

read in and stored in 2 ways, one as a list ResidualPhaseTensor object for

each matching station and the other in a structured array with all the important information. The structured array is the one that is used for plotting. It is computed each time plot() is called so if it is manipulated it is reset. The array is sorted by relative offset, so no special order of input is needed for the file names. However, the station names should be verbatim between surveys, otherwise it will not work.

The residual phase tensor is calculated as $I - (\Phi_2)^{-1} (\Phi_1)$

The default coloring is by the geometric mean as $\text{sqrt}(\Phi_{\min} * \Phi_{\max})$, which defines the percent change between measurements.

There are a lot of parameters to change how the plot looks, have a look below if you figure looks a little funny. The most useful will be xstretch, ystretch and ellipse_size

The ellipses are normalized by the largest Phi_max of the survey.

To get a list of .edi files that you want to plot -> :Example:

```
>>> import mtpy.imaging.mtplot as mtplot
>>> import os
>>> edipath1 = r"/home/EDIfiles1"
>>> edilist1 = [os.path.join(edipath1,edi) for edi in os.listdir(edipath1)
>>> ...      if edi.find('.edi')>0]
>>> edipath2 = r"/home/EDIfiles2"
>>> edilist2 = [os.path.join(edipath2,edi) for edi in os.listdir(edipath2)
>>> ...      if edi.find('.edi')>0]
>>> # color by phimin with a range of 0-5 deg
```

- If you want to plot geometric mean colored from white to orange in a range of 0 to 10 percent you can do it one of two ways->

1) :Example:

```
>>> edict = {'range':(0,10), 'cmap':'mt_wh2or',                               'colorby':
   ↵ 'geometric_mean', 'size':10}
>>> pt1 = mtplot.residual_pt_ps(edilist1, edilst2, ellipse_dict=edict)
```

2) :Example:

```
>>> pt1 = mtplot.residual_pt_ps(edilist1, edilst2, ellipse_dict=edict,           ↵
   ↵   plot_yn='n')
>>> pt1.ellipse_colorby = 'geometric_mean'
>>> pt1.ellipse_cmap = 'mt_wh2or'
>>> pt1.ellipse_range = (0, 10)
>>> pt1.ellipse_size = 10
>>> pt1.plot()
```

- If you want to save the plot as a pdf with a generic name ->

Example

```
:: >>> pt1.save_figure(r"/home/PTFigures", file_format='pdf', dpi=300) File saved to
   ↵ '/home/PTFigures/PTPseudoSection.pdf'
```

get_pt_color_array(*rpt_array*)

Get the appropriate color by array.

plot()

Plot residual phase tensor.

property rotation_angle

Rotation angle.

mtpy.imaging.plot_resphase_maps module**Description:**

Plots resistivity and phase maps for a given frequency

References:

CreationDate: 4/19/18 Developer: rakib.hassan@gov.au

Revision History:

LastUpdate: 4/19/18 RH
2022-09 JP

```
class mtpy.imaging.plot_resphase_maps.PlotResPhaseMaps(mt_data, **kwargs)
```

Bases: *PlotBaseMaps*

Plots apparent resistivity and phase in map view from a list of edi files.

Arguments:

```
**fn_list** : list of strings
              full paths to .edi files to plot

**fig_size** : tuple or list (x, y) in inches
              dimensions of the figure box in inches, this is a default
              unit of matplotlib. You can use this so make the plot
              fit the figure box to minimize spaces from the plot axes
              to the figure box. *default* is [8, 8]

**mapscale** : [ 'deg' | 'm' | 'km' ]
              Scale of the map coordinates.

                  * 'deg' --> degrees in latitude and longitude

                  * 'm' --> meters for easting and northing

                  * 'km' --> kilometers for easting and northing

**plot_yn** : [ 'y' | 'n' ]
              *'y'* to plot on creating an instance

              *'n'* to not plot on creating an instance

**title** : string
              figure title

**dpi** : int
              dots per inch of the resolution. *default* is 300

**font_size** : float
              size of the font that labels the plot, 2 will be added
              to this number for the axis labels.
```

property map_units

Map units.

plot()

Plot function.

mtpy.imaging.plot_spectrogram module**plotft**

*PlotTF -> will plot a time frequency distribution of your choice

Created on Mon Aug 19 16:24:29 2013

@author: jpeacock

class mtpy.imaging.plot_spectrogram.PlotTF(*time_series*, *tf_type='smethod'*, ***kwargs*)

Bases: object

Class to plot Time-Frequency.

plot()

Plot the time frequency distribution.

redraw_plot()

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

Example

```
:: >>> tf1.plot_type = 'tf' >>> tf1.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*.

Arguments:

```
**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/TF_tftype.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.
```

(continues on next page)

(continued from previous page)

```
* 'n' will leave plot open

:Example: ::

>>> # save plot as a jpg
>>> tf1.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 be on the major ticks and gray
>>> tf1.ax.grid(True, which='major', color=(.5,.5,.5))
>>> tf1.update_plot()
```

mtpy.imaging.plot_stations module**PlotStations**

Plots station locations in map view.

Created on Fri Jun 07 18:20:00 2013

@author: jpeacock-pr

class mtpy.imaging.plot_stations.PlotStations(*geo_df*, ***kwargs*)

Bases: *PlotBase*

Plot station locations in map view.

Uses contextily to get the basemap. See <https://contextily.readthedocs.io/en/latest/index.html> for more information about options.

plot()

Plot function. :param cx_source: DESCRIPTION providers.USGS.USTopo, defaults to cx. :type cx_source: TYPE, optional :param cx_zoom: DESCRIPTION, defaults to None. :type cx_zoom: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

mtpy.imaging.plot_strike module

Created on Thu May 30 18:28:24 2013

@author: jpeacock-pr

class mtpy.imaging.plot_strike.PlotStrike(*mt_data*, ***kwargs*)

Bases: *PlotBase*

PlotStrike will plot the strike estimated from the invariants, phase tensor

and the tipper in either a rose diagram or xy plot

plots the strike angle as determined by phase tensor azimuth (Caldwell et al. [2004]) and invariants of the impedance tensor (Weaver et al. [2003]).

The data is split into decades where the histogram for each is plotted in the form of a rose diagram with a range of 0 to 180 degrees. Where 0 is North and 90 is East. The median angle of the period band is set in polar diagram. The top row is the strike estimated from the invariants of the impedance tensor. The bottom

row is the azimuth estimated from the phase tensor. If tipper is ‘y’ then the 3rd row is the strike determined from the tipper, which is orthogonal to the induction arrow direction.

param fs

font size for labels of plotting. *Default* is 10

param rot_z

angle of rotation clockwise positive. *Default* is 0

param period_tolerance

float Tolerance level to match periods from different edi files. *Default* is 0.05

param text_pad

padding of the angle label at the bottom of each polar diagram. *Default* is 1.65

param text_size

font size

param plot_range

[‘data’ | (period_min,period_max)] period range to estimate the strike angle. Options are:

- ‘**data**’ for estimating the strike for all periods

in the data.

- (pmin,pmax) for period min and period max, input as (log10(pmin),log10(pmax))

param plot_type

[1 | 2] - 1 to plot individual decades in one plot - 2 to plot all period ranges into one polar diagram

for each strike angle estimation

param plot_tipper

[True | False] - True to plot the tipper strike - False to not plot tipper strike

param pt_error_floor

Maximum error in degrees that is allowed to estimate strike. *Default* is None allowing all estimates to be used.

param fold

[True | False] * True to plot only from 0 to 180 * False to plot from 0 to 360

param plot_orthogonal

[True | False] * True to plot the orthogonal strike directions * False to not

param color

[True | False] * True to plot shade colors * False to plot all in one color

param color_inv

color of invariants plots

param color_pt

color of phase tensor plots

param color_tip

color of tipper plots

param ring_spacing

spacing of rings in polar plots

param ring_limits

(min count, max count) set each plot have these limits

param plot_orientation
 [‘h’ | ‘v’] horizontal or vertical plots

Example

)

```
>>> #---Turn on Tipper
>>> strike.plot_tipper = True
>>> #---Plot only main directions
>>> strike.plot_orthogonal = False
>>> # Redraw plot
>>> strike.redraw_plot()
>>> # plot only from 0-180
>>> strike.fold = True
>>> strike.redraw_plot()
>>> #---save the plot---
>>> strike.save_plot(r"/home/Figures")
```

get_estimate(estimate, period_range=None)

Get estimate.

get_mean(estimate_df)

Get mean value.

get_median(estimate_df)

Get median value.

get_mode(estimate_df)

Get mode from a histrogram.

get_plot_array(estimate, period_range=None)

Get a plot array that has the min and max angles.

get_stats(estimate, period_range=None)

Print stats nicely.

make_strike_df()

Make strike array

Note

Polar plots assume the azimuth is an angle measured counterclockwise positive from x = 0. Therefore all angles are calculated as 90 - angle to make them conform to the polar plot convention..

plot(show=True)

Plot Strike angles as rose plots.

property rotation_angle

Rotation angle.

Module contents

```
class mtpy.imaging.PlotMTResponse(z_object=None, t_object=None, pt_obj=None, station='MT Response', **kwargs)
```

Bases: [PlotBase](#)

Plots Resistivity and phase for the different modes of the MT response.

At

the moment it supports the input of an .edi file. Other formats that will

be supported are the impedance tensor and errors with an array of periods and .j format.

The normal use is to input an .edi file, however it would seem that not everyone uses this format, so you can input the data and put it into arrays or objects like class mtpy.core.z.Z. Or if the data is in resistivity and phase format they can be input as arrays or a class mtpy.imaging.mtplot.ResPhase. Or you can put it into a class mtpy.imaging.mtplot.MTplot.

The plot places the apparent resistivity in log scale in the top panel(s), depending on the plot_num. The phase is below this, note that 180 degrees has been added to the yx phase so the xy and yx phases plot in the same quadrant. Both the resistivity and phase share the same x-axis which is in log period, short periods on the left to long periods on the right. So if you zoom in on the plot both plots will zoom in to the same x-coordinates. If there is tipper information, you can plot the tipper as a third panel at the bottom, and also shares the x-axis. The arrows are in the convention of pointing towards a conductor. The xx and yy components can be plotted as well, this adds two panels on the right. Here the phase is left unwrapped. Other parameters can be added as subplots such as strike, skew and phase tensor ellipses.

To manipulate the plot you can change any of the attributes listed below and call `redraw_plot()`. If you know more about matplotlib and want to change axes parameters, that can be done by changing the parameters in the axes attributes and then call `update_plot()`, note the plot must be open.

property period

Plot period.

plot()

`PlotResPhase(filename,fig_num)` will plot the apparent resistivity and phase for a single station.

property plot_model_error

Plot model error.

property rotation_angle

Rotation angle.

```
class mtpy.imaging.PlotMultipleResponses(mt_data, **kwargs)
```

Bases: [PlotBase](#)

Plots multiple MT responses simultaneously either in single plots or in one plot of sub-figures or in a single plot with subfigures for each component.

Arguments::

fn_list

[list of filenames to plot] ie. [fn_1, fn_2, ...], *default* is None

plot_num

[[1 | 2 | 3]]

- 1 for just Ex/By and Ey/Bx *default*
- 2 for all 4 components

- 3 for off diagonal plus the determinant

plot_style

`[['1' | 'all' | 'compare']]`

determines the plotting style:

- **'1'** for plotting each station in a different figure. *default*
- **'all'** for plotting each station in a subplot all in the same figure
- **'compare'** for comparing the responses all in one plot. Here the responses are colored from dark to light. This plot can get messy if too many stations are plotted.

plot()

Plot the apparent resistivity and phase.

property plot_model_error

Plot model error.

property rotation_angle

Rotation angle.

class mtpy.imaging.PlotPenetrationDepth1D(*tf*, *kwargs*)**

Bases: *PlotBase*

Plot the depth of penetration based on the Niblett-Bostick approximation.

property depth_units

Depth units.

plot()

Plot the depth of investigation as a 1d plot with period on the y-axis and depth on the x axis :return: DESCRIPTION. :rtype: TYPE

class mtpy.imaging.PlotPenetrationDepthMap(*mt_data*, *kwargs*)**

Bases: *PlotBaseMaps*

Plot the depth of penetration based on the Niblett-Bostick approximation.

property depth_units

Depth units.

plot()

Plot the depth of investigation as a 1d plot with period on the y-axis and depth on the x axis :return: DESCRIPTION. :rtype: TYPE

class mtpy.imaging.PlotPhaseTensor(*pt_object*, *station=None*, *kwargs*)**

Bases: *PlotBase*

Will plot phase tensor, strike angle, min and max phase angle, azimuth, skew, and ellipticity as subplots on one plot. It can plot the resistivity tensor along side the phase tensor for comparison.

plot(*rotation_angle=None*)

Plots the phase tensor elements.

property rotation_angle

Rotation angle.

```
class mtpy.imaging.PlotPhaseTensorMaps(mt_data, **kwargs)
Bases: PlotBaseMaps

Plots phase tensor ellipses in map view from a list of edi files.

property map_scale
    Map scale.

plot(fig=None, save_path=None, show=True, raster_file=None, raster_kwarg={})
    Plots the phase tensor map. :param raster_kwarg:
        Defaults to {}.
```

Parameters

- **raster_file** – Defaults to None.
- **fig** – Optional figure object, defaults to None.
- **save_path** – Path to folder for saving plots, defaults to None.
- **show** – Show plots if True, defaults to True.
- **raster_dict** – Dictionary containing information for a raster to be plotted below the phase tensors.

property rotation_angle

Rotation angle.

property skew_cmap_bounds

Skew bounds for a segmented colorbar.

```
class mtpy.imaging.PlotPhaseTensorPseudoSection(mt_data, **kwargs)
```

Bases: PlotBaseProfile

PlotPhaseTensorPseudoSection will plot the phase tensor ellipses in a pseudo section format

To get a list of .edi files that you want to plot -> :Example:

```
>>> import mtpy.imaging.mtplot as mtplot
>>> import os
>>> edipath = r"/home/EDIfiles"
>>> edilist = [os.path.join(edipath,edi) for edi in os.listdir(edipath)
>>> ...           if edi.find('.edi')>0]
```

- If you want to plot minimum phase colored from blue to red in a range of

20 to 70 degrees you can do it one of two ways->

1) :Example:

```
>>> edict = {'range':(20,70), 'cmap':'mt_b12gr2rd','colorby':'phimin'}
>>> pt1 = mtplot.plot_pt_pseudosection(fn_list=edilist,
                                         ellipse_dict=edict)
```

2) :Example:

```
>>> pt1 = mtplot.plot_pt_pseudosection(fn_list=edilist, plot_yn='n')
>>> pt1.ellipse_colorby = 'phimin'
>>> pt1.ellipse_cmap = 'mt_b12gr2rd'
>>> pt1.ellipse_range = (20,70)
>>> pt1.plot()
```

- If you want to add real induction arrows that are scaled by 10 and point away from a conductor ->

Example

```
>>> pt1.plot_tipper = 'yr'
>>> pt1.arrow_size = 10
>>> pt1.arrow_direction = -1
>>> pt1.redraw_plot()
```

- If you want to save the plot as a pdf with a generic name ->

Example

```
:: >>> pt1.savefig(r"/home/PTFigures", file_format='pdf', dpi=300) File saved to
'/home/PTFigures/PTPseudoSection.pdf'
```

plot()

Plots the phase tensor pseudo section.

See class doc string for

more details.

class mtpy.imaging.PlotResPhaseMaps(*mt_data*, *kwargs*)**

Bases: *PlotBaseMaps*

Plots apparent resistivity and phase in map view from a list of edi files.

Arguments:

```
**fn_list** : list of strings
              full paths to .edi files to plot

**fig_size** : tuple or list (x, y) in inches
              dimensions of the figure box in inches, this is a default
              unit of matplotlib. You can use this so make the plot
              fit the figure box to minimize spaces from the plot axes
              to the figure box. *default* is [8, 8]

**mapscale** : [ 'deg' | 'm' | 'km' ]
              Scale of the map coordinates.

              * 'deg' --> degrees in latitude and longitude

              * 'm' --> meters for easting and northing

              * 'km' --> kilometers for easting and northing
```

(continues on next page)

(continued from previous page)

```

**plot_yn** : [ 'y' | 'n' ]
    *'y'* to plot on creating an instance
    *'n'* to not plot on creating an instance

**title** : string
    figure title

**dpi** : int
    dots per inch of the resolution. *default* is 300

**font_size** : float
    size of the font that labels the plot, 2 will be added
    to this number for the axis labels.

```

property map_units

Map units.

plot()

Plot function.

```
class mtpy.imaging.PlotResPhasePseudoSection(mt_data, **kwargs)
```

Bases: *PlotBaseProfile*

Plot a resistivity and phase pseudo section for different components

Need to input one of the following lists::

plot()

Plot function. :return: DESCRIPTION. :rtype: TYPE

```
class mtpy.imaging.PlotResidualPTMaps(mt_data_01, mt_data_02, frequencies=array([1.0000000e-03,
    1.42510267e-03, 2.03091762e-03, 2.89426612e-03,
    4.12462638e-03, 5.87801607e-03, 8.37677640e-03,
    1.19377664e-02, 1.70125428e-02, 2.42446202e-02,
    3.45510729e-02, 4.92388263e-02, 7.01703829e-02,
    1.0000000e-01, 1.42510267e-01, 2.03091762e-01,
    2.89426612e-01, 4.12462638e-01, 5.87801607e-01,
    8.37677640e-01, 1.19377664e+00, 1.70125428e+00,
    2.42446202e+00, 3.45510729e+00, 4.92388263e+00,
    7.01703829e+00, 1.0000000e+01, 1.42510267e+01,
    2.03091762e+01, 2.89426612e+01, 4.12462638e+01,
    5.87801607e+01, 8.37677640e+01, 1.19377664e+02,
    1.70125428e+02, 2.42446202e+02, 3.45510729e+02,
    4.92388263e+02, 7.01703829e+02, 1.0000000e+03]),
    **kwargs)
```

Bases: *PlotBase*

This will plot residual phase tensors in a map for a single frequency.

The data is read in and stored in 2 ways, one as a list ResidualPhaseTensor object for each matching station and the other in a structured array with all the important information. The structured array is the one that is used for plotting. It is computed each time plot() is called so if it is manipulated it is reset. The array is sorted by relative offset, so no special order of input is needed for the file names. However, the station names should be verbatim between surveys, otherwise it will not work.

The residual phase tensor is calculated as $I - (\Phi_2)^{-1} (\Phi_1)$

The default coloring is by the geometric mean as $\sqrt{\Phi_{\min} \cdot \Phi_{\max}}$, which defines the percent change between measurements.

There are a lot of parameters to change how the plot looks, have a look below if you figure looks a little funny.
The most useful will be `ellipse_size`

The ellipses are normalized by the largest `Phi_max` of the survey.

Example

```
>>> import mtpy.imaging.mtplot as mtplot
>>> import os
>>> edipath1 = r"/home/EDIfiles1"
>>> edilist1 = [os.path.join(edipath1,edi) for edi in os.
->     .listdir(edipath1)
>>> ...      if edi.find('.edi')>0]
>>> edipath2 = r"/home/EDIfiles2"
>>> edilist2 = [os.path.join(edipath2,edi) for edi in os.
->     .listdir(edipath2)
>>> ...      if edi.find('.edi')>0]
>>> # color by phimin with a range of 0-5 deg
>>> ptmap = mtplot.plot_residual_pt_maps(edilist1, edilist2,
->      freqspot=10,
>>> ...                                     ellipse_dict={'size':1,
->      ...                                         'range':(0,5)})
>>>
>>> #
>>> #---add an image---
>>> ptmap.image_file = r"/home/Maps/Basemap.jpg"
>>> ptmap.image_extent = (0,10,0,10)
>>> ptmap.redraw_plot()
>>> #
>>> #---Save the plot---
>>> ptmap.save_plot(r"/home/EDIfiles",file_format='pdf')
>>> 'Saved figure to /home/EDIfiles/PTMaps/PTmap_phimin_10.0_Hz.pdf'
```

Example

```
>>> #change the axis label and grid color
>>> ptmap.ax.set_xlabel('Latitude (deg)')
>>> ptmap.ax.grid(which='major', color=(.5,1,0))
>>> ptmap.update_plot()
```

Example

```
>>> # plot seismic hypocenters from a file
>>> lat, lon, depth = np.loadtxt(r"/home/seismic_hypocenter.txt")
>>> ptmap.ax.scatter(lon, lat, marker='o')
```

property `map_scale`

Map scale.

`plot()`

Plot residual phase tensor.

property rotation_angle

Rotation angle.

```
class mtpy.imaging.PlotResidualPTPpseudoSection(mt_data_01, mt_data_02,
                                                    frequencies=array([1.00000000e-03, 1.42510267e-03,
                                                    2.03091762e-03, 2.89426612e-03, 4.12462638e-03,
                                                    5.87801607e-03, 8.37677640e-03, 1.19377664e-02,
                                                    1.70125428e-02, 2.42446202e-02, 3.45510729e-02,
                                                    4.92388263e-02, 7.01703829e-02, 1.00000000e-01,
                                                    1.42510267e-01, 2.03091762e-01, 2.89426612e-01,
                                                    4.12462638e-01, 5.87801607e-01, 8.37677640e-01,
                                                    1.19377664e+00, 1.70125428e+00, 2.42446202e+00,
                                                    3.45510729e+00, 4.92388263e+00, 7.01703829e+00,
                                                    1.00000000e+01, 1.42510267e+01, 2.03091762e+01,
                                                    2.89426612e+01, 4.12462638e+01, 5.87801607e+01,
                                                    8.37677640e+01, 1.19377664e+02, 1.70125428e+02,
                                                    2.42446202e+02, 3.45510729e+02, 4.92388263e+02,
                                                    7.01703829e+02, 1.00000000e+03]), **kwargs)
```

Bases: *PlotBaseProfile*

This will plot residual phase tensors in a pseudo section.

The data is

read in and stored in 2 ways, one as a list ResidualPhaseTensor object for

each matching station and the other in a structured array with all the important information. The structured array is the one that is used for plotting. It is computed each time plot() is called so if it is manipulated it is reset. The array is sorted by relative offset, so no special order of input is needed for the file names. However, the station names should be verbatim between surveys, otherwise it will not work.

The residual phase tensor is calculated as $I - (\Phi_2)^{-1}(\Phi_1)$

The default coloring is by the geometric mean as $\sqrt{\Phi_{\min} \cdot \Phi_{\max}}$, which defines the percent change between measurements.

There are a lot of parameters to change how the plot looks, have a look below if you figure looks a little funny. The most useful will be xstretch, ystretch and ellipse_size

The ellipses are normalized by the largest Phi_max of the survey.

To get a list of .edi files that you want to plot -> :Example:

```
>>> import mtpy.imaging.mtplot as mtplot
>>> import os
>>> edipath1 = r"/home/EDIfiles1"
>>> edilist1 = [os.path.join(edipath1, edi) for edi in os.listdir(edipath1)
>>> ...           if edi.find('.edi')>0]
>>> edipath2 = r"/home/EDIfiles2"
>>> edilist2 = [os.path.join(edipath2, edi) for edi in os.listdir(edipath2)
>>> ...           if edi.find('.edi')>0]
>>> # color by phimin with a range of 0-5 deg
```

- If you want to plot geometric mean colored from white to orange in a range of 0 to 10 percent you can do it one of two ways->

1) :Example:

```
>>> edict = {'range':(0,10), 'cmap':'mt_wh2or', 'colorby':  
    'geometric_mean', 'size':10}  
>>> pt1 = mtplot.residual_pt_ps(edilist1, edilst2, ellipse_dict=edict)
```

2) :Example:

```
>>> pt1 = mtplot.residual_pt_ps(edilist1, edilst2, ellipse_dict=edict,  
    plot_yn='n')  
>>> pt1.ellipse_colorby = 'geometric_mean'  
>>> pt1.ellipse_cmap = 'mt_wh2or'  
>>> pt1.ellipse_range = (0, 10)  
>>> pt1.ellipse_size = 10  
>>> pt1.plot()
```

- If you want to save the plot as a pdf with a generic name ->

Example

```
:: >>> pt1.save_figure(r"/home/PTFigures", file_format='pdf', dpi=300) File saved to  
'/home/PTFigures/PTPseudoSection.pdf'
```

get_pt_color_array(*rpt_array*)

Get the appropriate color by array.

plot()

Plot residual phase tensor.

property rotation_angle

Rotation angle.

class mtpy.imaging.PlotStations(*geo_df*, *kwargs*)**

Bases: *PlotBase*

Plot station locations in map view.

Uses contextily to get the basemap. See <https://contextily.readthedocs.io/en/latest/index.html> for more information about options.

plot()

Plot function. :param cx_source: DESCRIPTION providers.USGS.USTopo, defaults to cx. :type cx_source: TYPE, optional :param cx_zoom: DESCRIPTION, defaults to None. :type cx_zoom: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

class mtpy.imaging.PlotStrike(*mt_data*, *kwargs*)**

Bases: *PlotBase*

PlotStrike will plot the strike estimated from the invariants, phase tensor

and the tipper in either a rose diagram or xy plot

plots the strike angle as determined by phase tensor azimuth (Caldwell et al. [2004]) and invariants of the impedance tensor (Weaver et al. [2003]).

The data is split into decades where the histogram for each is plotted in the form of a rose diagram with a range of 0 to 180 degrees. Where 0 is North and 90 is East. The median angle of the period band is set in polar diagram. The top row is the strike estimated from the invariants of the impedance tensor. The bottom row is the azimuth estimated from the phase tensor. If tipper is 'y' then the 3rd row is the strike determined from the tipper, which is orthogonal to the induction arrow direction.

param fs

font size for labels of plotting. *Default* is 10

param rot_z

angle of rotation clockwise positive. *Default* is 0

param period_tolerance

float Tolerance level to match periods from different edi files. *Default* is 0.05

param text_pad

padding of the angle label at the bottom of each polar diagram. *Default* is 1.65

param text_size

font size

param plot_range

[‘data’ | (period_min,period_max)] period range to estimate the strike angle.
Options are:

- ‘**data**’ for estimating the strike for all periods
in the data.
- (pmin,pmax) for period min and period max, input as
(log10(pmin),log10(pmax))

param plot_type

[1 | 2] - 1 to plot individual decades in one plot - 2 to plot all period ranges into
one polar diagram

for each strike angle estimation

param plot_tipper

[True | False] - True to plot the tipper strike - False to not plot tipper strike

param pt_error_floor

Maximum error in degrees that is allowed to estimate strike. *Default* is None
allowing all estimates to be used.

param fold

[True | False] * True to plot only from 0 to 180 * False to plot from 0 to 360

param plot_orthogonal

[True | False] * True to plot the orthogonal strike directions * False to not

param color

[True | False] * True to plot shade colors * False to plot all in one color

param color_inv

color of invariants plots

param color_pt

color of phase tensor plots

param color_tip

color of tipper plots

param ring_spacing

spacing of rings in polar plots

param ring_limits

(min count, max count) set each plot have these limits

param plot_orientation

[‘h’ | ‘v’] horizontal or vertical plots

Example

)

```
>>> #---Turn on Tipper
>>> strike.plot_tipper = True
>>> #---Plot only main directions
>>> strike.plot_orthogonal = False
>>> # Redraw plot
>>> strike.redraw_plot()
>>> # plot only from 0-180
>>> strike.fold = True
>>> strike.redraw_plot()
>>> #---save the plot---
>>> strike.save_plot(r"/home/Figures")
```

get_estimate(*estimate*, *period_range*=None)

Get estimate.

get_mean(*estimate_df*)

Get mean value.

get_median(*estimate_df*)

Get median value.

get_mode(*estimate_df*)

Get mode from a histogram.

get_plot_array(*estimate*, *period_range*=None)

Get a plot array that has the min and max angles.

get_stats(*estimate*, *period_range*=None)

Print stats nicely.

make_strike_df()

Make strike array

Note

Polar plots assume the azimuth is an angle measured counterclockwise positive from $x = 0$. Therefore all angles are calculated as $90 - \text{angle}$ to make them conform to the polar plot convention..

plot(*show*=True)

Plot Strike angles as rose plots.

property rotation_angle

Rotation angle.

mtpy.modeling package

Subpackages

mtpy.modeling.modem package

Submodules

mtpy.modeling.modem.config module

ModEM

```
# Generate files for ModEM
# revised by JP 2017 # revised by AK 2017 to bring across functionality from ak branch
class mtpy.modeling.modem.config.ModEMConfig(**kwargs)
    Bases: object
    Read and write configuration files for how each inversion is run.

    add_dict(fn=None, obj=None)
        Add dictionary based on file name or object.

    write_config_file(save_dir=None, config_fn_basename='ModEM_inv.cfg')
        Write a config file based on provided information.
```

mtpy.modeling.modem.control_fwd module

ModEM

```
# Generate files for ModEM
# revised by JP 2017 # revised by AK 2017 to bring across functionality from ak branch # revised by OA 2024 to update
docs
class mtpy.modeling.modem.control_fwd.ControlFwd(**kwargs)
    Bases: object
    Reads and writes control files for ModEM to control how the forward solver starts and how it is run.

    num_qmr_iter Number of QMR iterations per divergence correction.
        int, default is 40

    max_num_div_calls Maximum number of divergence correction calls. int,
        default is 20

    max_num_div_iters Maximum number of divergence correction iterations.
        int, default is 100

    misfit_tol_fwd Misfit tolerance for EM forward solver. float,
        default is 1e-07

    misfit_tol_adj Misfit tolerance for EM adjoint solver. float,
        default is 1e-07

    misfit_tol_div Misfit tolerance for divergence correction. float,
        default is 1e-05

    save_path Path at which to save the control file. PosixPath,
        default is current working directory.

    fn_basename Name of the control file. str, default is
        'control.fwd'

    property control_fn
        Control fn.
```

read_control_file(control_fn=None)

Read in a control file.

write_control_file(control_fn=None, save_path=None, fn_basename=None)

Write control file.

Arguments::

control_fn

[string] full path to save control file to *default* is save_path/fn_basename

save_path

[string] directory path to save control file to *default* is cwd

fn_basename

[string] basename of control file *default* is control.inv

mtpy.modeling.modem.control_inv module

ModEM

Generate files for ModEM

revised by JP 2017 # revised by AK 2017 to bring across functionality from ak branch # revised by OA 2024 to update docs

class mtpy.modeling.modem.control_inv.ControlInv(kwargs)**

Bases: object

Reads and writes control files for ModEM to control how the inversion starts and how it is run.

output_fn Model and data output file name “prefix phrase”, i.e.

written to the front of ModEM output file names before the iteration number and file extension. str, *default* is ‘MODULAR_NLCG’

lambda_initial Initial damping factor lambda. int, *default* is 10 **lambda_step** To update lambda, divide by this value. int, *default*

is 10

model_search_step Initial search step in model units. int, *default* is

1

rms_reset_search Restart when rms diff is less than this value. float,

default is 0.002

rms_target Exit search when rms is less than this value. float,

default is 1.05

lambda_exit Exit when lambda is less than this value. float,

default is 0.0001

max_iterations Maximum number of iterations. int, *default* is 100 **save_path** Path at which to save the control file. PosixPath,

default is current working directory

fn_basename Name of the control file. str, *default* is

‘control.inv’

```
property control_fn
    Control fn.

read_control_file(control_fn=None)
    Read in a control file.

write_control_file(control_fn=None, save_path=None, fn_basename=None)
    Write control file.
```

Arguments::

```
control_fn
    [string] full path to save control file to default is save_path/fn_basename

save_path
    [string] directory path to save control file to default is cwd

fn_basename
    [string] basename of control file default is control.inv
```

mtpy.modeling.modem.convariance module**ModEM**

```
# Generate files for ModEM
```

```
# revised by JP 2017 # revised by AK 2017 to bring across functionality from ak branch
```

```
class mtpy.modeling.modem.convariance(grid_dimensions=None, **kwargs)
```

Bases: object

Class that deals with model covariance and smoothing, writing covariance (.cov) files for use in ModEM.

grid_dimensions Grid dimensions excluding air layers (Nx, Ny, NzEarth) smoothing_east Smoothing in east direction. float, *default* is 0.3 smoothing_north Smoothing in north direction. float, *default* is 0.3 smoothing_z Smoothing in vertical direction. float, *default* is

0.3

smoothing_num Number of smoothing iterations. int, *default* is 1 exception_list List of exceptions. list, *default* is empty mask_arr Mask array. numpy.ndarray, *default* is None save_path Path at which to save the covariance file. PosixPath,

default is current working directory

fn_basename Name of the covariance file. *default* is
‘covariance.cov’

```
property cov_fn
```

Cov fn.

```
get_parameters()
```

Get parameters.

```
read_cov_file(cov_fn)
```

Reads a ModEM covariance (.cov) file. :param cov_fn: Filename of the target ModEM covariance (.cov) file. :type cov_fn: str :raises CovarianceError: Error related to the covariance class.

```
write_cov_vtk_file(cov vtk_fn, model_fn=None, grid_east=None, grid_north=None, grid_z=None)
```

Write a vtk file of the covariance to match things up.

```
write_covariance_file(cov_fn=None, save_path=None, fn_basename=None, res_model=None,
                      sea_water=0.3, air=1000000000000.0)
```

Writes a ModEM covariance (.cov) file. :param cov_fn: Name for the covariance file. *default* is None, defaults to None. :type cov_fn: str | Path, optional :param save_path: Location at which to save the covariance file.

default is None, defaults to None.

Parameters

- **fn_basename** (_type_, optional) – _description_, *default* is None, defaults to None.
- **res_model** (_type_, optional) – Resistivity model the covariance should be generated from, *default* is None, defaults to None.
- **sea_water** (float, optional) – Electrical resistivity of sea water cells in Ohm-meters within the model domain, *default* is 0.3, defaults to 0.3.
- **air** (float, optional) – Electrical resistivity of air cells in Ohm-meters within the model domain, *default* is 1.0e12, defaults to 1.0e12.

Raises

CovarianceError – Error related to the covariance class.

mtpy.modeling.modem.data module

ModEM

```
# Generate files for ModEM
```

```
# revised by JP 2017 # revised by AK 2017 to bring across functionality from ak branch # revised by JP 2021 adding
functionality and updating. # revised by JP 2022 to work with new structure of a central object
```

```
class mtpy.modeling.modem.data.Data(dataframe=None, center_point=None, **kwargs)
```

Bases: object

Data will read and write .dat files for ModEM and convert a WS data file to ModEM format.

..note: :: the data is interpolated onto the given periods such that all

stations invert for the same periods. The interpolation is a linear interpolation of each of the real and imaginary parts of the impedance tensor and induction tensor. See mtpy.core.mt.MT.interpolate for more details

Parameters

- ****kwargs** –
- **center_point** – Defaults to None.
- **dataframe** – Defaults to None.
- **edi_list** – List of edi files to read.

property data_filename

Data filename.

property dataframe

Dataframe function.

fix_data_file(*fn=None*, *n=3*)

A newer compiled version of Modem outputs slightly different headers This aims to convert that into the older format :param fn: DESCRIPTION, defaults to None. :type fn: TYPE, optional :param n: DESCRIPTION, defaults to 3. :type n: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

get_n_stations(*mode*)

Get n stations.

property model_parameters

Model parameters.

property n_periods

N periods.

property period

Period function.

read_data_file(*data_fn*)

Read data file. :param data_fn: Full path to data file name. :type data_fn: string or Path :raises ValueError: If cannot compute component.

write_data_file(*file_name=None*, *save_path=None*, *fn_basename=None*, *elevation=False*)

Write data file. :param file_name:

Defaults to None.

Parameters

- **save_path** (string or Path, optional) – Full directory to save file to, defaults to None.
- **fn_basename** (string, optional) – Basename of the saved file, defaults to None.
- **elevation** (boolean, optional) – If True adds in elevation from ‘rel_elev’ column in data array, defaults to False.

Raises

- **NotImplementedError** – If the inversion mode is not supported.
- **ValueError** – mtpy.utils.exceptions.ValueError if a parameter.

Returns

Full path to data file.

Return type

Path

mtpy.modeling.modem.exception module**ModEM**

```
# Generate files for ModEM
# revised by JP 2017 # revised by AK 2017 to bring across functionality from ak branch
exception mtpy.modeling.modem.exception.DataError
Bases: ModEMError
Raise for ModEM Data class specific exceptions.
```

```
exception mtpy.modeling.modem.exception.ModEMError
```

Bases: Exception

mtpy.modeling.modem.model module

ModEM

Generate files for ModEM

revised by JP 2017 # revised by AK 2017 to bring across functionality from ak branch # revised by JP 2021 updating functionality and updating docs

```
class mtpy.modeling.modem.model.Model(station_locations=None, center_point=None, **kwargs)
```

Bases: object

Make and read a FE mesh grid

The mesh assumes the coordinate system where:

x == North y == East z == + down

All dimensions are in meters.

The mesh is created by first making a regular grid around the station area, then padding cells are added that exponentially increase to the given extensions. Depth cell increase on a log10 scale to the desired depth, then padding cells are added that increase exponentially..

Parameters

station_object	-	mtpy.modeling.modem.Stations object ..	seealso::
mtpy.modeling.modem.Stations			

Examples

Example 1 → create mesh first then data file

```
>>> import mtpy.modeling.modem as modem
>>> import os
>>> # 1) make a list of all .edi files that will be inverted for
>>> edi_path = r"/home/EDI_Files"
>>> edi_list = [os.path.join(edi_path, edi)
```

for edi in os.listdir(edi_path)

```
>>> ... if edi.find('.edi') > 0]
>>> # 2) Make a Stations object
>>> stations_obj = modem.Stations()
>>> stations_obj.get_station_locations_from_edi(edi_list)
>>> # 3) make a grid from the stations themselves with 200m cell
spacing
>>> mmesh = modem.Model(station_obj)
>>> # change cell sizes
>>> mmesh.cell_size_east = 200,
>>> mmesh.cell_size_north = 200
>>> mmesh.ns_ext = 300000 # north-south extension
>>> mmesh.ew_ext = 200000 # east-west extension of model
>>> mmesh.make_mesh()
>>> # check to see if the mesh is what you think it should be
```

(continues on next page)

(continued from previous page)

```
>>> msmesh.plot_mesh()
>>> # all is good write the mesh file
>>> msmesh.write_model_file(save_path=r"/home/modem/Inv1")
>>> # create data file
>>> md = modem.Data(edi_list, station_locations=mmesh.station_
>>> locations)
>>> md.write_data_file(save_path=r"/home/modem/Inv1")
```

Example 2 → Rotate Mesh

```
>>> mmesh.mesh_rotation_angle = 60
>>> mmesh.make_mesh()
```

Note

ModEM assumes all coordinates are relative to North and East, and does not accommodate mesh rotations, therefore, here the rotation is of the stations, which essentially does the same thing. You will need to rotate your data to align with the ‘new’ coordinate system.

Attributes	Description
_logger	python logging object that put messages in logging format defined in logging configure file, see MtPyLog more information
cell_number_ew	optional for user to specify the total number of cells on the east-west direction. <i>default</i> is None
cell_number_ns	optional for user to specify the total number of cells on the north-south direction. <i>default</i> is None
cell_size_east	mesh block width in east direction <i>default</i> is 500
cell_size_north	mesh block width in north direction <i>default</i> is 500
grid_center	center of the mesh grid
grid_east	overall distance of grid nodes in east direction
grid_north	overall distance of grid nodes in north direction
grid_z	overall distance of grid nodes in z direction
model_fn	full path to initial file name
model_fn_basename	default name for the model file name
n_air_layers	number of air layers in the model. <i>default</i> is 0
n_layers	total number of vertical layers in model
nodes_east	relative distance between nodes in east direction
nodes_north	relative distance between nodes in north direction
nodes_z	relative distance between nodes in z direction
pad_east	number of cells for padding on E and W sides <i>default</i> is 7
pad_north	number of cells for padding on S and N sides <i>default</i> is 7
pad_num	number of cells with cell_size with outside of station area. <i>default</i> is 3
pad_method	method to use to create padding: extent1, extent2 - calculate based on ew_ext and ns_ext stretch - calculate based on pad_stretch factors

continues on next page

Table 1 – continued from previous page

Attributes	Description
pad_stretch_h	multiplicative number for padding in horizontal direction.
pad_stretch_v	padding cells N & S will be pad_root_north** <i>(x)</i>
pad_z	number of cells for padding at bottom <i>default</i> is 4
ew_ext	E-W extension of model in meters
ns_ext	N-S extension of model in meters
res_scale	scaling method of res, supports ‘log’ - for log e format ‘log’ or ‘log10’ - for log with base 10 ‘linear’ - linear scale <i>default</i> is ‘log’
res_list	list of resistivity values for starting model
res_model	starting resistivity model
res_initial_value	resistivity initial value for the resistivity model <i>default</i> is 100
mesh_rotation_angle	Angle to rotate the grid to. Angle is measured positive clockwise assuming North is 0 and east is 90. <i>default</i> is None
save_path	path to save file to
sea_level	sea level in grid_z coordinates. <i>default</i> is 0
station_locations	location of stations
title	title in initial file
z1_layer	first layer thickness
z_bottom	absolute bottom of the model <i>default</i> is 300,000
z_target_depth	Depth of deepest target, <i>default</i> is 50,000

add_layers_to_mesh(*n_add_layers=None*, *layer_thickness=None*, *where='top'*)

Function to add constant thickness layers to the top or bottom of mesh.

Note: It is assumed these layers are added before the topography. If you want to add topography layers, use function `add_topography_to_model` :param *n_add_layers*: Integer, number of layers to add, defaults to None. :param *layer_thickness*: Real value or list/array. Thickness of layers,

Can provide a single value

or a list/array containing multiple layer thicknesses, defaults to None.

Parameters

where – Where to add, top or bottom, defaults to “top”.

add_topography_from_data(*interp_method='nearest'*, *air_resistivity=1000000000000.0*, *topography_buffer=None*, *airlayer_type='log_up'*)

Wrapper around `add_topography_to_model` that allows creating a surface model from EDI data. The Data grid and station elevations will be used to make a ‘surface’ tuple that will be passed to `add_topography_to_model` so a surface model can be interpolated from it.

The surface tuple is of format (lon, lat, elev) containing station locations. :param *data_object*: A ModEm data

object that has been filled with data from EDI files.

Parameters

- **interp_method** (*str, optional*) – Same as add_topography_to_model, defaults to “nearest”.
- **air_resistivity** (*float, optional*) – Same as add_topography_to_model, defaults to 1e12.
- **topography_buffer** (*float, optional*) – Same as add_topography_to_model, defaults to None.
- **airlayer_type** (*str, optional*) – Same as add_topography_to_model, defaults to “log_up”.

```
add_topography_to_model(topography_file=None, surface=None, topography_array=None,
                       interp_method='nearest', air_resistivity=1000000000000.0,
                       topography_buffer=None, airlayer_type='log_up', max_elev=None,
                       shift_east=0, shift_north=0)
```

If air_layers is non-zero, will add topo: read in topograph file, make a surface model.

Call project_stations_on_topography in the end, which will re-write the .dat file.

If n_airlayers is zero, then cannot add topo data, only bathymetry is needed. :param shift_north:

Defaults to 0.

Parameters

- **shift_east** – Defaults to 0.
- **max_elev** – Defaults to None.
- **surface** – Defaults to None.
- **topography_file** – File containing topography (arcgis ascii grid), defaults to None.
- **topography_array** – Alternative to topography_file - array of elevation values on model grid, defaults to None.
- **interp_method** – Interpolation method for topography, ‘nearest’, ‘linear’, or ‘cubic’, defaults to “nearest”.
- **air_resistivity** – Resistivity value to assign to air, defaults to 1e12.
- **topography_buffer** – Buffer around stations to calculate minimum and maximum topography value to use for meshing, defaults to None.
- **airlayer_type** – How to set air layer thickness - options are ‘constant’ for constant air layer thickness, or ‘log’, for logarithmically increasing air layer thickness upward, defaults to “log_up”.

```
assign_resistivity_from_surface_data(top_surface, bottom_surface, resistivity_value)
```

Assign resistivity value to all points above or below a surface requires the surface_dict attribute to exist and contain data for surface key (can get this information from ascii file using project_surface)

inputs surface_name = name of surface (must correspond to key in surface_dict) resistivity_value = value to assign where = ‘above’ or ‘below’ - assign resistivity above or below the

surface

```
interpolate_elevation(surface_file=None, surface=None, get_surface_name=False, method='nearest',
                      fast=True, shift_north=0, shift_east=0)
```

Project a surface to the model grid and add resulting elevation data to a dictionary called surface_dict. Assumes the surface is in lat/long coordinates (wgs84)

returns nothing returned, but surface data are added to surface_dict under the key given by surface_name.
inputs choose to provide either surface_file (path to file) or surface (tuple). If both are provided then surface tuple takes priority.

surface elevations are positive up, and relative to sea level. surface file format is:

```
ncols 3601 nrows 3601 xllcorner -119.00013888889 (longitude of lower left) yllcorner 36.999861111111  
(latitude of lower left) cellsize 0.0002777777777778 NODATA_value -9999 elevation data W -> E N |  
V S
```

Alternatively, provide a tuple with: (lon,lat,elevation) where elevation is a 2D array (shape (ny,nx)) containing elevation points (order S -> N, W -> E) and lon, lat are either 1D arrays containing list of longitudes and latitudes (in the case of a regular grid) or 2D arrays with same shape as elevation array containing longitude and latitude of each point.

other inputs: surface_epsg = epsg number of input surface, default is 4326 for lat/lon(wgs84) method = interpolation method. Default is ‘nearest’, if model grid is dense compared to surface points then choose ‘linear’ or ‘cubic’

make_mesh(*verbose=True***)**

Create finite element mesh according to user-input parameters.

The mesh is built by:

1. Making a regular grid within the station area.
2. Adding pad_num of cell_width cells outside of station area
3. Adding padding cells to given extension and number of padding cells.
4. Making vertical cells starting with z1_layer increasing logarithmically (base 10) to z_target_depth and num_layers.
5. Add vertical padding cells to desired extension.
6. Check to make sure none of the stations lie on a node. If they do then move the node by .02*cell_width

make_z_mesh(*n_layers=None*)

New version of make_z_mesh. make_z_mesh and M.

property model_epsg

Model epsg.

property model_fn

Model fn.

property model_parameters

Get important model parameters to write to a file for documentation later.

property nodes_east

Nodes east.

property nodes_north

Nodes north.

property nodes_z

Nodes z.

property plot_east

Plot east.

```
plot_mesh(**kwargs)
    Plot model mesh. :param **kwargs: :param plot_topography: DESCRIPTION, defaults to False. :type
    plot_topography: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

property plot_north
    Plot north.

property plot_z
    Plot z.

read_gocad_sgrid_file(sgrid_header_file, air_resistivity=1e+39, sea_resistivity=0.3,
sgrid_positive_up=True)
    Read a gocad sgrid file and put this info into a ModEM file.

    Note: can only deal with grids oriented N-S or E-W at this stage, with orthogonal coordinates

read_model_file(model_fn=None)
    Read an initial file and return the pertinent information including grid positions in coordinates relative to
    the center point (0,0) and starting model.

    Note that the way the model file is output, it seems is that the blocks are setup as

ModEM: WS::
    0—> N_north 0—>N_east ||| V V N_east N_north

    Arguments:
    **model_fn** : full path to initializing file.

    Outputs:
    **nodes_north** : np.array(nx)
        array of nodes in S --> N direction

    **nodes_east** : np.array(ny)
        array of nodes in the W --> E direction

    **nodes_z** : np.array(nz)
        array of nodes in vertical direction positive downwards

    **res_model** : dictionary
        dictionary of the starting model with keys as layers

    **res_list** : list
        list of resistivity values in the model

    **title** : string
        title string

property save_path
    Save path.

write_geosoft_xyz(save_fn, c_east=0, c_north=0, c_z=0, pad_north=0, pad_east=0, pad_z=0)
    Write an XYZ file readable by Geosoft

    All input units are in meters.. :param save_fn: Full path to save file to. :type save_fn: string or Path :param
    c_east: Center point in the east direction, defaults to 0. :type c_east: float, optional :param c_north: Center
    point in the north direction, defaults to 0. :type c_north: float, optional :param c_z: Center point elevation,
```

defaults to 0. :type c_z: float, optional :param pad_north: Number of cells to cut from the north-south edges, defaults to 0. :type pad_north: int, optional :param pad_east: Number of cells to cut from the east-west edges, defaults to 0. :type pad_east: int, optional :param pad_z: Number of cells to cut from the bottom, defaults to 0. :type pad_z: int, optional

`write_gocad_sgrid_file(fn=None, origin=[0, 0, 0], clip=0, no_data_value=-99999)`

Write a model to gocad sgrid

optional inputs:

fn = filename to save to. File extension ('.sg') will be appended.

default is the model name with extension removed

origin = real world [x,y,z] location of zero point in model grid clip = how much padding to clip off the edge of the model for export,

provide one integer value or list of 3 integers for x,y,z directions

no_data_value = no data value to put in sgrid.

`write_model_file(**kwargs)`

Will write an initial file for ModEM.

Note that x is assumed to be S → N, y is assumed to be W → E and z is positive downwards. This means that index [0, 0, 0] is the southwest corner of the first layer. Therefore if you build a model by hand the layer block will look as it should in map view.

Also, the xgrid, ygrid and zgrid are assumed to be the relative distance between neighboring nodes. This is needed because wsinv3d builds the model from the bottom SW corner assuming the cell width from the init file.

Key Word Arguments:

```
**nodes_north** : np.array(nx)
    block dimensions (m) in the N-S direction.
    **Note** that the code reads the grid assuming that
    index=0 is the southern most point.

**nodes_east** : np.array(ny)
    block dimensions (m) in the E-W direction.
    **Note** that the code reads in the grid assuming that
    index=0 is the western most point.

**nodes_z** : np.array(nz)
    block dimensions (m) in the vertical direction.
    This is positive downwards.

**save_path** : string
    Path to where the initial file will be saved
    to save_path/model_fn_basename

**model_fn_basename** : string
    basename to save file to
    *default* is ModEM_Model.ws
    file is saved at save_path/model_fn_basename

**title** : string
    Title that goes into the first line
```

(continues on next page)

(continued from previous page)

```

*default* is Model File written by MTpy.modeling.modem

**res_model** : np.array((nx,ny,nz))
    Prior resistivity model.

    .. note:: again that the modeling code
    assumes that the first row it reads in is the southern
    most row and the first column it reads in is the
    western most column. Similarly, the first plane it
    reads in is the Earth's surface.

**res_starting_value** : float
    starting model resistivity value,
    assumes a half space in Ohm-m
    *default* is 100 Ohm-m

**res_scale** : [ 'loge' | 'log' | 'log10' | 'linear' ]
    scale of resistivity. In the ModEM code it
    converts everything to Loge,
    *default* is 'loge'

```

write_out_file(*save_fn*, *geographic_east*, *geographic_north*, *geographic_elevation*)

Will write an .out file for LeapFrog.

Note that y is assumed to be S → N, e is assumed to be W → E and z is positive upwards. This means that index [0, 0, 0] is the southwest corner of the first layer. :param *save_fn*: Full path to save file to. :type *save_fn*: string or Path :param *geographic_east*: Geographic center in easting (meters). :type *geographic_east*: float :param *geographic_north*: Geographic center in northing (meters). :type *geographic_north*: float :param *geographic_elevation*: Elevation of geographic center (meters). :type *geographic_elevation*: float :return: DESCRIPTION. :rtype: TYPE

write_ubb_files(*basename*, *c_east*=0, *c_north*=0, *c_z*=0)

Write a UBC .msh and .mod file. :param *basename*: :param *save_fn*: DESCRIPTION. :type *save_fn*: TYPE :param *c_east*: DESCRIPTION, defaults to 0. :type *c_east*: TYPE, optional :param *c_north*: DESCRIPTION, defaults to 0. :type *c_north*: TYPE, optional :param *c_z*: DESCRIPTION, defaults to 0. :type *c_z*: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

write_vtk_file(*vtk_save_path*=None, *vtk_fn_basename*='ModEM_model_res', *shift_east*=0, *shift_north*=0, *shift_z*=0, *units*='km', *coordinate_system*='nez+', *label*='resistivity')

Write a VTK file to plot in 3D rendering programs like Paraview. :param *label*:

Defaults to "resistivity".

Parameters

- **coordinate_system** – Defaults to "nez+".
- **units** – Defaults to "km".
- **shift_z** – Defaults to 0.
- **shift_north** – Defaults to 0.
- **shift_east** – Defaults to 0.
- **vtk_save_path**(string or Path, optional) – Directory to save vtk file to, defaults to None.

- **vtk_fn_basename** – Filename basename of vtk file, note that .vtr, defaults to “ModEM_model_res”.

Returns

Full path to VTK file.

Return type

Path

write_xyres(*save_path=None, location_type='EN', origin=[0, 0], model_epsg=None, depth_index='all', outfile_basename='DepthSlice', log_res=False, clip=[0, 0]*)

Write files containing depth slice data (x, y, res for each depth)

origin = x,y coordinate of zero point of ModEM_grid, or name of file

containing this info (full path or relative to model files)

save_path = path to save to, default is the model object save path *location_type* = ‘EN’ or ‘LL’ xy points saved as eastings/northings or

longitude/latitude, if ‘LL’ need to also provide *model_epsg*

model_epsg = epsg number that was used to project the model *outfile_basename* = string for basename for saving the depth slices. *log_res* = True/False - option to save resistivity values as log10

instead of linear

clip = number of cells to clip on each of the east/west and north/south edges.

write_xyzres(*savefile=None, location_type='EN', origin=[0, 0], model_epsg=None, log_res=False, model_utm_zone=None, clip=[0, 0]*)

Save a model file as a space delimited x y z res file.

mtpy.modeling.modem.residual module

ModEM

residuals class to contain RMS information

revised by JP 2017 revised by AK 2017 to bring across functionality from ak branch

class mtpy.modeling.modem.Residual(kwargs)**

Bases: *Data*

Class to contain residuals for each data point, and rms values for each station

Attributes/Key Words	Description
work_dir residual_fn residual_array	full path to data file numpy.ndarray (num_stations) structured to store data. keys are: <ul style="list-style-type: none">• station → station name• lat → latitude in decimal degrees• lon → longitude in decimal degrees• elev → elevation (m)• rel_east → relative east location to center_position (m)• rel_north → relative north location to center_position (m)• east → UTM east (m)• north → UTM north (m)• zone → UTM zone• z → impedance tensor residual (measured - modelled) (num_freq, 2, 2)• z_err → impedance tensor error array with shape (num_freq, 2, 2)• tip → Tipper residual (measured - modelled) (num_freq, 1, 2)• tiperr → Tipper array with shape (num_freq, 1, 2)
rms rms_array	numpy.ndarray structured to store station location values and rms. Keys are: <ul style="list-style-type: none">• station → station name• east → UTM east (m)• north → UTM north (m)• lat → latitude in decimal degrees• lon → longitude in decimal degrees• elev → elevation (m)• zone → UTM zone• rel_east → relative east location to center_position (m)• rel_north → relative north location to center_position (m)• rms → root-mean-square residual for each station
rms_tip rms_z	

calculate_rms()

Add columns for rms. :return: DESCRIPTION. :rtype: TYPE

plot_rms(kwargs)**

Plot RMS in different views. :param **kwargs: DESCRIPTION. :type **kwargs: TYPE :return: DESCRIPTION. :rtype: TYPE

plot_rms_per_period(*plot_type='all'*, ***kwargs*)

Plot rms per period. :param plot_type:

Defaults to “all”.

Parameters

****kwargs** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

read_residual_file(*residual_fn*)

Read residual file. :param residual_fn: DESCRIPTION, defaults to None. :type residual_fn: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

property rms_per_period_all

RMS per period.

property rms_per_period_per_component

RMS per period by component. :return: DESCRIPTION. :rtype: TYPE

mtpy.modeling.modem.station module

ModEM

Generate files for ModEM

revised by JP 2017 # revised by AK 2017 to bring across functionality from ak branch

class mtpy.modeling.modem.station.Stations(kwargs)**

Bases: object

Station locations class.

calculate_rel_locations(*shift_east=0*, *shift_north=0*)

Put station in a coordinate system relative to (shift_east, shift_north) (+) shift right or up (-) shift left or down

property center_point

Calculate the center point from the given station locations.

property east

East function.

property elev

Elev function.

get_station_locations(*input_list*)

Get station locations from a list of edi files.

Parameters

****input_list**** – list list of edi file names, or mt_objects

property lat

Lat function.

property lon

Lon function.

property model_epsg

Model epsg.

property model_utm_zone

Model utm zone.

property north

North function.

property rel_east

Rel east.

property rel_elev

Rel elev.

property rel_north

Rel north.

rotate_stations(*rotation_angle*)

Rotate stations assuming N is 0.

Parameters

****rotation_angle**** – float angle in degrees assuming N is 0

property station

Station function.

to_csv(*csv_fn*, *epsg=None*, *default_epsg=4326*, *geometry=False*)

Write a shape file of the station locations using geopandas which only takes in epsg numbers :param geometry:

Defaults to False.

Parameters

- **csv_fn**
- **shp_fn (string)** – Full path to new shapefile.
- **epsg (integer, defaults to None, optional)** – EPSG number to project to, defaults to None.
- **default_epsg** – The default EPSG number that the stations are, defaults to 4326.

to_geopd(*epsg=None*, *default_epsg=4326*)

Create a geopandas dataframe. :param epsg: EPSG number to project to, defaults to None. :type epsg: integer, defaults to None, optional :param default_epsg: The default EPSG number that the stations are, defaults to 4326.

to_shp(*shp_fn*, *epsg=None*, *default_epsg=4326*)

Write a shape file of the station locations using geopandas which only takes in epsg numbers :param shp_fn: Full path to new shapefile. :type shp_fn: string :param epsg: EPSG number to project to, defaults to None. :type epsg: integer, defaults to None, optional :param default_epsg: The default EPSG number that the stations are, defaults to 4326.

property utm_zone

Utm zone.

Module contents

class `mtpy.modeling.modem.ControlFwd(**kwargs)`

Bases: `object`

Reads and writes control files for ModEM to control how the forward solver starts and how it is run.

num_qmr_iter Number of QMR iterations per divergence correction.

int, *default* is 40

max_num_div_calls Maximum number of divergence correction calls. int, *default* is 20

max_num_div_iters Maximum number of divergence correction iterations. int, *default* is 100

misfit_tol_fwd Misfit tolerance for EM forward solver. float, *default* is 1e-07

misfit_tol_adj Misfit tolerance for EM adjoint solver. float, *default* is 1e-07

misfit_tol_div Misfit tolerance for divergence correction. float, *default* is 1e-05

save_path Path at which to save the control file. PosixPath, *default* is current working directory.

fn_basename Name of the control file. str, *default* is ‘control.fwd’,

property control_fn

Control fn.

read_control_file(*control_fn=None*)

Read in a control file.

write_control_file(*control_fn=None*, *save_path=None*, *fn_basename=None*)

Write control file.

Arguments::

control_fn

[string] full path to save control file to *default* is `save_path/fn_basename`

save_path

[string] directory path to save control file to *default* is cwd

fn_basename

[string] basename of control file *default* is `control.inv`

class `mtpy.modeling.modem.ControlInv(**kwargs)`

Bases: `object`

Reads and writes control files for ModEM to control how the inversion starts and how it is run.

output_fn Model and data output file name “prefix phrase”, i.e.

written to the front of ModEM output file names before the iteration number and file extension. str, *default* is ‘MODULAR_NLCG’

`lambda_initial` Initial damping factor lambda. int, *default* is 10 `lambda_step` To update lambda, divide by this value. int, *default*

is 10

model_search_step Initial search step in model units. int, *default* is

1

rms_reset_search Restart when rms diff is less than this value. float,

default is 0.002

rms_target Exit search when rms is less than this value. float,

default is 1.05

lambda_exit Exit when lambda is less than this value. float,

default is 0.0001

max_iterations Maximum number of iterations. int, *default* is 100
save_path Path at which to save the control file. PosixPath,

default is current working directory

fn_basename Name of the control file. str, *default* is

‘control.inv’

property control_fn

Control fn.

read_control_file(control_fn=None)

Read in a control file.

write_control_file(control_fn=None, save_path=None, fn_basename=None)

Write control file.

Arguments:

control_fn

[string] full path to save control file to *default* is save_path/fn_basename

save_path

[string] directory path to save control file to *default* is cwd

fn_basename

[string] basename of control file *default* is control.inv

class mtpy.modeling.modem.Covariance(grid_dimensions=None, **kwargs)

Bases: object

Class that deals with model covariance and smoothing, writing covariance (.cov) files for use in ModEM.

grid_dimensions Grid dimensions excluding air layers (Nx, Ny, NzEarth) smoothing_east Smoothing in east direction. float, *default* is 0.3 smoothing_north Smoothing in north direction. float, *default* is 0.3 smoothing_z Smoothing in vertical direction. float, *default* is

0.3

smoothing_num Number of smoothing iterations. int, *default* is 1
exception_list List of exceptions. list, *default* is empty
mask_arr Mask array. numpy.ndarray, *default* is None
save_path Path at which to save the covariance file. PosixPath,

default is current working directory

fn_basename Name of the covariance file. *default* is

‘covariance.cov’

```
property cov_fn
    Cov fn.

get_parameters()
    Get parameters.

read_cov_file(cov_fn)
    Reads a ModEM covariance (.cov) file. :param cov_fn: Filename of the target ModEM covariance (.cov) file. :type cov_fn: str :raises CovarianceError: Error related to the covariance class.

write_cov_vtk_file(cov vtk_fn, model_fn=None, grid_east=None, grid_north=None, grid_z=None)
    Write a vtk file of the covariance to match things up.

write_covariance_file(cov_fn=None, save_path=None, fn_basename=None, res_model=None, sea_water=0.3, air=1000000000000.0)
    Writes a ModEM covariance (.cov) file. :param cov_fn: Name for the covariance file. default is None, defaults to None. :type cov_fn: str | Path, optional :param save_path: Location at which to save the covariance file.

    default is None, defaults to None.

Parameters

- fn_basename (_type_, optional) – _description_, default is None, defaults to None.
- res_model (_type_, optional) – Resistivity model the covariance should be generated from, default is None, defaults to None.
- sea_water (float, optional) – Electrical resistivity of sea water cells in Ohm-meters within the model domain, default is 0.3, defaults to 0.3.
- air(float, optional) – Electrical resistivity of air cells in Ohm-meters within the model domain, default is 1.0e12, defaults to 1.0e12.

```

Raises**CovarianceError** – Error related to the covariance class.**class mtpy.modeling.modem.Data(dataframe=None, center_point=None, **kwargs)**

Bases: object

Data will read and write .dat files for ModEM and convert a WS data file to ModEM format.

..note: :: the data is interpolated onto the given periods such that all

stations invert for the same periods. The interpolation is a linear interpolation of each of the real and imaginary parts of the impedance tensor and induction tensor. See mtpy.core.mt.MT.interpolate for more details

Parameters

- ****kwargs** –
- **center_point** – Defaults to None.
- **dataframe** – Defaults to None.
- **edi_list** – List of edi files to read.

property data_filename

Data filename.

property dataframe

Dataframe function.

fix_data_file(*fn=None, n=3*)

A newer compiled version of Modem outputs slightly different headers This aims to convert that into the older format :param fn: DESCRIPTION, defaults to None. :type fn: TYPE, optional :param n: DESCRIPTION, defaults to 3. :type n: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

get_n_stations(*mode*)

Get n stations.

property model_parameters

Model parameters.

property n_periods

N periods.

property period

Period function.

read_data_file(*data_fn*)

Read data file. :param data_fn: Full path to data file name. :type data_fn: string or Path :raises ValueError: If cannot compute component.

write_data_file(*file_name=None, save_path=None, fn_basename=None, elevation=False*)

Write data file. :param file_name:

Defaults to None.

Parameters

- **save_path** (string or Path, optional) – Full directory to save file to, defaults to None.
- **fn_basename** (string, optional) – Basename of the saved file, defaults to None.
- **elevation** (boolean, optional) – If True adds in elevation from ‘rel_elev’ column in data array, defaults to False.

Raises

- **NotImplementedError** – If the inversion mode is not supported.
- **ValueError** – mtpy.utils.exceptions.ValueError if a parameter.

Returns

Full path to data file.

Return type

Path

exception mtpy.modeling.modem.DataError

Bases: *ModEMError*

Raise for ModEM Data class specific exceptions.

class mtpy.modeling.modem.ModEMConfig(kwargs)**

Bases: object

Read and write configuration files for how each inversion is run.

add_dict(fn=None, obj=None)

Add dictionary based on file name or object.

write_config_file(save_dir=None, config_fn_basename='ModEM_inv.cfg')

Write a config file based on provided information.

exception mtpy.modeling.modem.ModEMError

Bases: Exception

class mtpy.modeling.modem.Model(station_locations=None, center_point=None, **kwargs)

Bases: object

Make and read a FE mesh grid

The mesh assumes the coordinate system where:

x == North y == East z == + down

All dimensions are in meters.

The mesh is created by first making a regular grid around the station area, then padding cells are added that exponentially increase to the given extensions. Depth cell increase on a log10 scale to the desired depth, then padding cells are added that increase exponentially..

Parameters

****station_object**** – mtpy.modeling.modem.Stations object .. seealso::
mtpy.modeling.modem.Stations

Examples

Example 1 –> create mesh first then data file

```
>>> import mtpy.modeling.modem as modem
>>> import os
>>> # 1) make a list of all .edi files that will be inverted for
>>> edi_path = r"/home/EDI_Files"
>>> edi_list = [os.path.join(edi_path, edi)
```

for edi in os.listdir(edi_path)

```
>>> ...      if edi.find('.edi') > 0]
>>> # 2) Make a Stations object
>>> stations_obj = modem.Stations()
>>> stations_obj.get_station_locations_from_edi(edi_list)
>>> # 3) make a grid from the stations themselves with 200m cell_
->spacing
>>> mmesh = modem.Model(station_obj)
>>> # change cell sizes
>>> mmesh.cell_size_east = 200,
>>> mmesh.cell_size_north = 200
>>> mmesh.ns_ext = 300000 # north-south extension
>>> mmesh.ew_ext = 200000 # east-west extension of model
>>> mmesh.make_mesh()
>>> # check to see if the mesh is what you think it should be
>>> msmesh.plot_mesh()
>>> # all is good write the mesh file
>>> msmesh.write_model_file(save_path=r"/home/modem/Inv1")
```

(continues on next page)

(continued from previous page)

```
>>> # create data file
>>> md = modem.Data(edi_list, station_locations=mmesh.station_
    ↪locations)
>>> md.write_data_file(save_path=r"/home/modem/Inv1")
```

Example 2 → Rotate Mesh

```
>>> mmesh.mesh_rotation_angle = 60
>>> mmesh.make_mesh()
```

i Note

ModEM assumes all coordinates are relative to North and East, and does not accommodate mesh rotations, therefore, here the rotation is of the stations, which essentially does the same thing. You will need to rotate your data to align with the ‘new’ coordinate system.

Attributes	Description
_logger	python logging object that put messages in logging format defined in logging configure file, see MtPyLog more information
cell_number_ew	optional for user to specify the total number of cells on the east-west direction. <i>default</i> is None
cell_number_ns	optional for user to specify the total number of cells on the north-south direction. <i>default</i> is None
cell_size_east	mesh block width in east direction <i>default</i> is 500
cell_size_north	mesh block width in north direction <i>default</i> is 500
grid_center	center of the mesh grid
grid_east	overall distance of grid nodes in east direction
grid_north	overall distance of grid nodes in north direction
grid_z	overall distance of grid nodes in z direction
model_fn	full path to initial file name
model_fn_basename	default name for the model file name
n_air_layers	number of air layers in the model. <i>default</i> is 0
n_layers	total number of vertical layers in model
nodes_east	relative distance between nodes in east direction
nodes_north	relative distance between nodes in north direction
nodes_z	relative distance between nodes in z direction
pad_east	number of cells for padding on E and W sides <i>default</i> is 7
pad_north	number of cells for padding on S and N sides <i>default</i> is 7
pad_num	number of cells with cell_size with outside of station area. <i>default</i> is 3
pad_method	method to use to create padding: extent1, extent2 - calculate based on ew_ext and ns_ext stretch - calculate based on pad_stretch factors
pad_stretch_h	multiplicative number for padding in horizontal direction.

continues on next page

Table 2 – continued from previous page

Attributes	Description
pad_stretch_v	padding cells N & S will be pad_root_north** <i>(x)</i>
pad_z	number of cells for padding at bottom <i>default</i> is 4
ew_ext	E-W extension of model in meters
ns_ext	N-S extension of model in meters
res_scale	scaling method of res, supports ‘log’ - for log e format ‘log’ or ‘log10’ - for log with base 10 ‘linear’ - linear scale <i>default</i> is ‘log’
res_list	list of resistivity values for starting model
res_model	starting resistivity model
res_initial_value	resistivity initial value for the resistivity model <i>default</i> is 100
mesh_rotation_angle	Angle to rotate the grid to. Angle is measured positive clockwise assuming North is 0 and east is 90. <i>default</i> is None
save_path	path to save file to
sea_level	sea level in grid_z coordinates. <i>default</i> is 0
station_locations	location of stations
title	title in initial file
z1_layer	first layer thickness
z_bottom	absolute bottom of the model <i>default</i> is 300,000
z_target_depth	Depth of deepest target, <i>default</i> is 50,000

add_layers_to_mesh(*n_add_layers=None*, *layer_thickness=None*, *where='top'*)

Function to add constant thickness layers to the top or bottom of mesh.

Note: It is assumed these layers are added before the topography. If you want to add topography layers, use function add_topography_to_model :param *n_add_layers*: Integer, number of layers to add, defaults to None. :param *layer_thickness*: Real value or list/array. Thickness of layers,

Can provide a single value

or a list/array containing multiple layer thicknesses, defaults to None.

Parameters

where – Where to add, top or bottom, defaults to “top”.

add_topography_from_data(*interp_method='nearest'*, *air_resistivity=1000000000000.0*, *topography_buffer=None*, *airlayer_type='log_up'*)

Wrapper around add_topography_to_model that allows creating a surface model from EDI data. The Data grid and station elevations will be used to make a ‘surface’ tuple that will be passed to add_topography_to_model so a surface model can be interpolated from it.

The surface tuple is of format (lon, lat, elev) containing station locations. :param *data_object*: A ModEm data

object that has been filled with data from EDI files.

Parameters

- **interp_method** (*str, optional*) – Same as add_topography_to_model, defaults to “nearest”.

- **air_resistivity** (*float, optional*) – Same as add_topography_to_model, defaults to 1e12.
- **topography_buffer** (*float, optional*) – Same as add_topography_to_model, defaults to None.
- **airlayer_type** (*str, optional*) – Same as add_topography_to_model, defaults to “log_up”.

```
add_topography_to_model(topography_file=None, surface=None, topography_array=None,
                        interp_method='nearest', air_resistivity=1000000000000.0,
                        topography_buffer=None, airlayer_type='log_up', max_elev=None,
                        shift_east=0, shift_north=0)
```

If air_layers is non-zero, will add topo: read in topograph file, make a surface model.

Call project_stations_on_topography in the end, which will re-write the .dat file.

If n_airlayers is zero, then cannot add topo data, only bathymetry is needed. :param shift_north:

Defaults to 0.

Parameters

- **shift_east** – Defaults to 0.
- **max_elev** – Defaults to None.
- **surface** – Defaults to None.
- **topography_file** – File containing topography (arcgis ascii grid), defaults to None.
- **topography_array** – Alternative to topography_file - array of elevation values on model grid, defaults to None.
- **interp_method** – Interpolation method for topography, ‘nearest’, ‘linear’, or ‘cubic’, defaults to “nearest”.
- **air_resistivity** – Resistivity value to assign to air, defaults to 1e12.
- **topography_buffer** – Buffer around stations to calculate minimum and maximum topography value to use for meshing, defaults to None.
- **airlayer_type** – How to set air layer thickness - options are ‘constant’ for constant air layer thickness, or ‘log’, for logarithmically increasing air layer thickness upward, defaults to “log_up”.

```
assign_resistivity_from_surface_data(top_surface, bottom_surface, resistivity_value)
```

Assign resistivity value to all points above or below a surface requires the surface_dict attribute to exist and contain data for surface key (can get this information from ascii file using project_surface)

inputs surface_name = name of surface (must correspond to key in surface_dict) resistivity_value = value to assign where = ‘above’ or ‘below’ - assign resistivity above or below the

surface

```
interpolate_elevation(surface_file=None, surface=None, get_surface_name=False, method='nearest',
                      fast=True, shift_north=0, shift_east=0)
```

Project a surface to the model grid and add resulting elevation data to a dictionary called surface_dict. Assumes the surface is in lat/long coordinates (wgs84)

returns nothing returned, but surface data are added to surface_dict under the key given by surface_name.

inputs choose to provide either surface_file (path to file) or surface (tuple). If both are provided then surface tuple takes priority.

surface elevations are positive up, and relative to sea level. surface file format is:

```
ncols 3601 nrows 3601 xllcorner -119.00013888889 (longitude of lower left) yllcorner 36.999861111111  
(latitude of lower left) cellsize 0.00027777777777778 NODATA_value -9999 elevation data W -> E N |  
V S
```

Alternatively, provide a tuple with: (lon,lat,elevation) where elevation is a 2D array (shape (ny,nx)) containing elevation points (order S -> N, W -> E) and lon, lat are either 1D arrays containing list of longitudes and latitudes (in the case of a regular grid) or 2D arrays with same shape as elevation array containing longitude and latitude of each point.

other inputs: surface_epsg = epsg number of input surface, default is 4326 for lat/lon(wgs84) method = interpolation method. Default is ‘nearest’, if model grid is dense compared to surface points then choose ‘linear’ or ‘cubic’

make_mesh(*verbose=True***)**

Create finite element mesh according to user-input parameters.

The mesh is built by:

1. Making a regular grid within the station area.
2. Adding pad_num of cell_width cells outside of station area
3. Adding padding cells to given extension and number of padding cells.
4. Making vertical cells starting with z1_layer increasing logarithmically (base 10) to z_target_depth and num_layers.
5. Add vertical padding cells to desired extension.
6. Check to make sure none of the stations lie on a node. If they do then move the node by .02*cell_width

make_z_mesh(*n_layers=None*)

New version of make_z_mesh. make_z_mesh and M.

property model_epsg

Model epsg.

property model_fn

Model fn.

property model_parameters

Get important model parameters to write to a file for documentation later.

property nodes_east

Nodes east.

property nodes_north

Nodes north.

property nodes_z

Nodes z.

property plot_east

Plot east.

```
plot_mesh(**kwargs)
    Plot model mesh. :param **kwargs: :param plot_topography: DESCRIPTION, defaults to False. :type
    plot_topography: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

property plot_north
    Plot north.

property plot_z
    Plot z.

read_gocad_sgrid_file(sgrid_header_file, air_resistivity=1e+39, sea_resistivity=0.3,
sgrid_positive_up=True)
    Read a gocad sgrid file and put this info into a ModEM file.

    Note: can only deal with grids oriented N-S or E-W at this stage, with orthogonal coordinates

read_model_file(model_fn=None)
    Read an initial file and return the pertinent information including grid positions in coordinates relative to
    the center point (0,0) and starting model.

    Note that the way the model file is output, it seems is that the blocks are setup as

ModEM: WS::
    0—> N_north 0——>N_east | | | V V N_east N_north

    Arguments:
    **model_fn** : full path to initializing file.

    Outputs:
    **nodes_north** : np.array(nx)
        array of nodes in S --> N direction

    **nodes_east** : np.array(ny)
        array of nodes in the W --> E direction

    **nodes_z** : np.array(nz)
        array of nodes in vertical direction positive downwards

    **res_model** : dictionary
        dictionary of the starting model with keys as layers

    **res_list** : list
        list of resistivity values in the model

    **title** : string
        title string

property save_path
    Save path.

write_geosoft_xyz(save_fn, c_east=0, c_north=0, c_z=0, pad_north=0, pad_east=0, pad_z=0)
    Write an XYZ file readable by Geosoft

    All input units are in meters.. :param save_fn: Full path to save file to. :type save_fn: string or Path :param
    c_east: Center point in the east direction, defaults to 0. :type c_east: float, optional :param c_north: Center
    point in the north direction, defaults to 0. :type c_north: float, optional :param c_z: Center point elevation,
```

defaults to 0. :type c_z: float, optional :param pad_north: Number of cells to cut from the north-south edges, defaults to 0. :type pad_north: int, optional :param pad_east: Number of cells to cut from the east-west edges, defaults to 0. :type pad_east: int, optional :param pad_z: Number of cells to cut from the bottom, defaults to 0. :type pad_z: int, optional

write_gocad_sgrid_file(fn=None, origin=[0, 0, 0], clip=0, no_data_value=-99999)

Write a model to gocad sgrid

optional inputs:

fn = filename to save to. File extension ('.sg') will be appended.

default is the model name with extension removed

origin = real world [x,y,z] location of zero point in model grid clip = how much padding to clip off the edge of the model for export,

provide one integer value or list of 3 integers for x,y,z directions

no_data_value = no data value to put in sgrid.

write_model_file(kwargs)**

Will write an initial file for ModEM.

Note that x is assumed to be S → N, y is assumed to be W → E and z is positive downwards. This means that index [0, 0, 0] is the southwest corner of the first layer. Therefore if you build a model by hand the layer block will look as it should in map view.

Also, the xgrid, ygrid and zgrid are assumed to be the relative distance between neighboring nodes. This is needed because wsinv3d builds the model from the bottom SW corner assuming the cell width from the init file.

Key Word Arguments:

```
**nodes_north** : np.array(nx)
    block dimensions (m) in the N-S direction.
    **Note** that the code reads the grid assuming that
    index=0 is the southern most point.

**nodes_east** : np.array(ny)
    block dimensions (m) in the E-W direction.
    **Note** that the code reads in the grid assuming that
    index=0 is the western most point.

**nodes_z** : np.array(nz)
    block dimensions (m) in the vertical direction.
    This is positive downwards.

**save_path** : string
    Path to where the initial file will be saved
    to save_path/model_fn_basename

**model_fn_basename** : string
    basename to save file to
    *default* is ModEM_Model.ws
    file is saved at save_path/model_fn_basename

**title** : string
    Title that goes into the first line
```

(continues on next page)

(continued from previous page)

```

*default* is Model File written by MTpy.modeling.modem

**res_model** : np.array((nx,ny,nz))
    Prior resistivity model.

    .. note:: again that the modeling code
    assumes that the first row it reads in is the southern
    most row and the first column it reads in is the
    western most column. Similarly, the first plane it
    reads in is the Earth's surface.

**res_starting_value** : float
    starting model resistivity value,
    assumes a half space in Ohm-m
    *default* is 100 Ohm-m

**res_scale** : [ 'loge' | 'log' | 'log10' | 'linear' ]
    scale of resistivity. In the ModEM code it
    converts everything to Loge,
    *default* is 'loge'

```

write_out_file(*save_fn*, *geographic_east*, *geographic_north*, *geographic_elevation*)

Will write an .out file for LeapFrog.

Note that y is assumed to be S → N, e is assumed to be W → E and z is positive upwards. This means that index [0, 0, 0] is the southwest corner of the first layer. :param *save_fn*: Full path to save file to. :type *save_fn*: string or Path :param *geographic_east*: Geographic center in easting (meters). :type *geographic_east*: float :param *geographic_north*: Geographic center in northing (meters). :type *geographic_north*: float :param *geographic_elevation*: Elevation of geographic center (meters). :type *geographic_elevation*: float :return: DESCRIPTION. :rtype: TYPE

write_ubb_files(*basename*, *c_east*=0, *c_north*=0, *c_z*=0)

Write a UBC .msh and .mod file. :param *basename*: :param *save_fn*: DESCRIPTION. :type *save_fn*: TYPE :param *c_east*: DESCRIPTION, defaults to 0. :type *c_east*: TYPE, optional :param *c_north*: DESCRIPTION, defaults to 0. :type *c_north*: TYPE, optional :param *c_z*: DESCRIPTION, defaults to 0. :type *c_z*: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

write_vtk_file(*vtk_save_path*=None, *vtk_fn_basename*='ModEM_model_res', *shift_east*=0, *shift_north*=0, *shift_z*=0, *units*='km', *coordinate_system*='nez+', *label*='resistivity')

Write a VTK file to plot in 3D rendering programs like Paraview. :param *label*:

Defaults to "resistivity".

Parameters

- **coordinate_system** – Defaults to "nez+".
- **units** – Defaults to "km".
- **shift_z** – Defaults to 0.
- **shift_north** – Defaults to 0.
- **shift_east** – Defaults to 0.
- **vtk_save_path** (string or Path, optional) – Directory to save vtk file to, defaults to None.

- **vtk_fn_basename** – Filename basename of vtk file, note that .vtr, defaults to “ModEM_model_res”.

Returns

Full path to VTK file.

Return type

Path

write_xyres(*save_path=None, location_type='EN', origin=[0, 0], model_epsg=None, depth_index='all', outfile_basename='DepthSlice', log_res=False, clip=[0, 0]*)

Write files containing depth slice data (x, y, res for each depth)

origin = x,y coordinate of zero point of ModEM_grid, or name of file

containing this info (full path or relative to model files)

save_path = path to save to, default is the model object save path *location_type* = ‘EN’ or ‘LL’ xy points saved as eastings/northings or

longitude/latitude, if ‘LL’ need to also provide *model_epsg*

model_epsg = epsg number that was used to project the model *outfile_basename* = string for basename for saving the depth slices. *log_res* = True/False - option to save resistivity values as log10

instead of linear

clip = number of cells to clip on each of the east/west and north/south edges.

write_xyzres(*savefile=None, location_type='EN', origin=[0, 0], model_epsg=None, log_res=False, model_utm_zone=None, clip=[0, 0]*)

Save a model file as a space delimited x y z res file.

class `mtpy.modeling.modem.Residual(**kwargs)`

Bases: `Data`

Class to contain residuals for each data point, and rms values for each station

Attributes/Key Words	Description
work_dir residual_fn residual_array	full path to data file numpy.ndarray (num_stations) structured to store data. keys are: <ul style="list-style-type: none">• station → station name• lat → latitude in decimal degrees• lon → longitude in decimal degrees• elev → elevation (m)• rel_east → relative east location to center_position (m)• rel_north → relative north location to center_position (m)• east → UTM east (m)• north → UTM north (m)• zone → UTM zone• z → impedance tensor residual (measured - modelled) (num_freq, 2, 2)• z_err → impedance tensor error array with shape (num_freq, 2, 2)• tip → Tipper residual (measured - modelled) (num_freq, 1, 2)• tiperr → Tipper array with shape (num_freq, 1, 2)
rms rms_array	numpy.ndarray structured to store station location values and rms. Keys are: <ul style="list-style-type: none">• station → station name• east → UTM east (m)• north → UTM north (m)• lat → latitude in decimal degrees• lon → longitude in decimal degrees• elev → elevation (m)• zone → UTM zone• rel_east → relative east location to center_position (m)• rel_north → relative north location to center_position (m)• rms → root-mean-square residual for each station
rms_tip rms_z	

calculate_rms()

Add columns for rms. :return: DESCRIPTION. :rtype: TYPE

plot_rms(kwargs)**

Plot RMS in different views. :param **kwargs: DESCRIPTION. :type **kwargs: TYPE :return: DESCRIPTION. :rtype: TYPE

plot_rms_per_period(*plot_type='all'*, ***kwargs*)

Plot rms per period. :param plot_type:

Defaults to “all”.

Parameters

****kwargs** – DESCRIPTION.

Returns

DESCRIPTION.

Return type

TYPE

read_residual_file(*residual_fn*)

Read residual file. :param residual_fn: DESCRIPTION, defaults to None. :type residual_fn: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

property rms_per_period_all

RMS per period.

property rms_per_period_per_component

RMS per period by component. :return: DESCRIPTION. :rtype: TYPE

mtpy.modeling.occam1d package

Submodules

mtpy.modeling.occam1d.data module

Created on Mon Oct 30 13:31:30 2023

@author: jpeacock

class mtpy.modeling.occam1d.data.Occam1DDData(*mt_dataframe*, ***kwargs*)

Bases: object

reads and writes occam 1D data files

Attributes	Description
<code>_data_fn</code>	basename of data file <i>default</i> is Occam1DDDataFile
<code>_header_line</code>	header line for description of data columns
<code>_ss</code>	string spacing <i>default</i> is 6*' '
<code>_string_fmt</code>	format of data <i>default</i> is '+.6e'
<code>data</code>	array of data
<code>data_fn</code>	full path to data file
<code>freq</code>	frequency array of data
<code>mode</code>	mode to invert for [‘TE’ ‘TM’ ‘det’]
<code>phase_te</code>	array of TE phase
<code>phase_tm</code>	array of TM phase
<code>res_te</code>	array of TE apparent resistivity
<code>res_tm</code>	array of TM apparent resistivity
<code>resp_fn</code>	full path to response file
<code>save_path</code>	path to save files to

Methods	Description
write_data_file	write an Occam1D data file
read_data_file	read an Occam1D data file
read_resp_file	read a .resp file output by Occam1D

Example

```
>>> import mtpy.modeling.occam1d as occam1d
>>> #--> make a data file for TE mode
>>> d1 = occam1d.Data()
>>> d1.write_data_file(edi_file=r'/home/MT/mt01.edi', res_err=10, ↴
    phase_err=2.5,
>>> ...                                save_path=r"/home/occam1d/mt01/TE", mode='TE')
```

property mode
property mode_01
property mode_02
read_data_file(data_fn)
 reads a 1D data file

Arguments:

data_fn : full path to data file

Returns:

Occam1D.rpdct : dictionary with keys:
 '*freq*' : an array of frequencies with length nf
 '*resxy*'
 [TE resistivity array with shape (nf,4) for (0) data,]
 (1) dataerr, (2) model, (3) modelerr
 '*resyx*'
 [TM resistivity array with shape (nf,4) for (0) data,]
 (1) dataerr, (2) model, (3) modelerr
 '*phasexy*'
 [TE phase array with shape (nf,4) for (0) data,]
 (1) dataerr, (2) model, (3) modelerr
 '*phaseyx*'
 [TM phase array with shape (nf,4) for (0) data,]
 (1) dataerr, (2) model, (3) modelerr

Example

```
>>> old = occam1d.Data()
>>> old.data_fn = r"/home/Occam1D/Line1/Inv1_TE/MT01TE.dat"
>>> old.read_data_file()
```

read_resp_file(*resp_fn=None*, *data_fn=None*)

read response file

resp_fn : full path to response file

data_fn : full path to data file

freq : an array of frequencies with length nf

res_te

[TE resistivity array with shape (nf,4) for (0) data,]

(1) dataerr, (2) model, (3) modelerr

res_tm

[TM resistivity array with shape (nf,4) for (0) data,]

(1) dataerr, (2) model, (3) modelerr

phase_te

[TE phase array with shape (nf,4) for (0) data,]

(1) dataerr, (2) model, (3) modelerr

phase_tm

[TM phase array with shape (nf,4) for (0) data,]

(1) dataerr, (2) model, (3) modelerr

Example

```
::           >>> old      =      occam1d.Data()      >>>      old.data_fn      =
r"/home/occam1d/mt01/TE/Occam1D_DataFile_TE.dat"                      >>>
old.read_resp_file(r"/home/occam1d/mt01/TE/TE_7.resp")
```

write_data_file(*filename*, *mode='det'*, *remove_outofquadrant=False*)

make1Ddatafile will write a data file for Occam1D

Arguments:**rp_tuple**

[np.ndarray (freq, res, res_err, phase, phase_err)] with res, phase having shape (num_freq, 2, 2).

edi_file

[string] full path to edi file to be modeled.

save_path

[string] path to save the file, if None set to dirname of station if edipath = None. Otherwise set to dirname of edipath.

thetar

[float] rotation angle to rotate Z. Clockwise positive and N=0 *default = 0*

mode

[['te' | 'tm' | 'det']]

mode to model can be (*default*='both'):

- ‘te’ for just TE mode (res/phase)
- ‘tm’ for just TM mode (res/phase)
- ‘det’ **for the determinant of Z (converted to res/phase)**

add ‘z’ to any of these options to model impedance tensor values instead of res/phase

res_err

[float] errorbar for resistivity values. Can be set to (*default* = ‘data’):

- ‘data’ for errorbars from the data
- percent number ex. 10 for ten percent

phase_err

[float] errorbar for phase values. Can be set to (*default* = ‘data’):

- ‘data’ for errorbars from the data
- percent number ex. 10 for ten percent

res_errorfloor: float

error floor for resistivity values in percent

phase_errorfloor: float

error floor for phase in degrees

remove_outofquadrant: True/False; option to remove the resistivity and

phase values for points with phases out of the 1st/3rd quadrant (occam requires 0 < phase < 90 degrees; phases in the 3rd quadrant are shifted to the first by adding 180 degrees)

Example

```
>>> import mtpy.modeling.occam1d as occam1d
>>> #--> make a data file
>>> d1 = occam1d.Data()
>>> d1.write_data_file(edi_file=r'/home/MT/mt01.edi', res_err=10,
>>> ...                         phase_err=2.5, mode='TE',
>>> ...                         save_path=r"/home/occam1d/mt01/TE")
```

mtpy.modeling.occam1d.model module

Created on Mon Oct 30 13:29:22 2023

@author: jpeacock

class mtpy.modeling.occam1d.model.Occam1DModel(*model_fn=None*, ***kwargs*)

Bases: object

read and write the model file fo Occam1D

All depth measurements are in meters.

Attributes	Description
_model_fn	basename for model file <i>default</i> is Model1D
_ss	string spacing in model file <i>default</i> is 3*' '
_string_fmt	format of model layers <i>default</i> is '.0f'
air_layer_height	height of air layer <i>default</i> is 10000
bottom_layer	bottom of the model <i>default</i> is 50000
itdict	dictionary of values from iteration file
iter_fn	full path to iteration file
model_depth	array of model depths
model_fn	full path to model file
model_penalty	array of penalties for each model layer
model_preference_penalty	array of model preference penalties for each layer
model_prefernce	array of preferences for each layer
model_res	array of resistivities for each layer
n_layers	number of layers in the model
num_params	number of parameters to invert for (n_layers+2)
pad_z	padding of model at depth <i>default</i> is 5 blocks
save_path	path to save files
target_depth	depth of target to investigate
z1_layer	depth of first layer <i>default</i> is 10

Methods	Description
write_model_file	write an Occam1D model file, where depth increases on a logarithmic scale
read_model_file	read an Occam1D model file
read_iter_file	read an .iter file output by Occam1D

Example

```
>>> #--> make a model file
>>> m1 = occam1d.Model()
>>> m1.write_model_file(save_path=r"/home/occam1d/mt01/TE")
```

read_iter_file(iter_fn=None, model_fn=None)

read an 1D iteration file

Arguments:

imode : mode to read from

Returns:

Occam1D.itdict : dictionary with keys of the header:

model_res

[fills this array with the appropriate] values (0) for data, (1) for model

Example

```
>>> m1 = occam1d.Model()
>>> m1.model_fn = r"/home/occam1d/mt01/TE/Model1D"
>>> m1.read_iter_file(r"/home/Occam1D/Inv1_TE/M01TE_15.iter")
```

read_model_file(model_fn=None)

will read in model 1D file

Arguments:

modelfn : full path to model file

Fills attributes:

- model_depth' : depth of model in meters
- model_res : value of resistivity
- model_penalty : penalty
- model_preference : preference
- model_penalty_preference : preference penalty

Example

```
>>> m1 = occam1d.Model()
>>> m1.savepath = r"/home/Occam1D/Line1/Inv1_TE"
>>> m1.read_model_file()
```

write_model_file(save_path=None, **kwargs)

Makes a 1D model file for Occam1D.

Arguments:

save_path : path to save file to, if just path saved as
savepathmodel.mod, if None defaults to dirpath

n_layers : number of layers

bottom_layer : depth of bottom layer in meters

target_depth : depth to target under investigation

pad_z : padding on bottom of model past target_depth

z1_layer : depth of first layer in meters

air_layer_height : height of air layers in meters

Returns:

Occam1D.modelfn = full path to model file

..Note: This needs to be redone.

Example

```
>>> old = occam.Occam1D()
>>> old.make1DModelFile(savepath=r"/home/Occam1D/Line1/Inv1_TE",
>>>                               nlayers=50, bottomlayer=10000, z1layer=50)
>>> Wrote Model file: /home/Occam1D/Line1/Inv1_TE/Model1D
```

mtpy.modeling.occam1d.plot_l2 module

Created on Mon Oct 30 13:34:53 2023

@author: jpeacock

class mtpy.modeling.occam1d.plot_l2.PlotOCCAM1DL2(*dir_path*, *model_fn*, ***kwargs*)

Bases: *PlotBase*

Plot L2 curve of iteration vs rms and roughness.

Arguments::

- rms_arr**
[structured array with keys:]
- ‘iteration’ –> for iteration number (int)
 - ‘rms’ –> for rms (float)
 - ‘roughness’ –> for roughness (float)

Keywords/attributes	Description
ax1	matplotlib.axes instance for rms vs iteration
ax2	matplotlib.axes instance for roughness vs rms
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 axes tick labels, axes labels is +2
plot_yn	[‘y’ ‘n’] ‘y’ –> to plot on instantiation ‘n’ –> to not plot on instantiation
rms_arr	structure np.array as described above
rms_color	color of rms marker and line
rms_lw	line width of rms line
rms_marker	marker for rms values
rms_marker_size	size of marker for rms values
rms_mean_color	color of mean line
rms_median_color	color of median line
rough_color	color of roughness line and marker
rough_font_size	font size for iteration number inside roughness marker
rough_lw	line width for roughness line
rough_marker	marker for roughness
rough_marker_size	size of marker for roughness
subplot_bottom	subplot spacing from bottom
subplot_left	subplot spacing from left
subplot_right	subplot spacing from right
subplot_top	subplot spacing from top

plot()

Plot L2 curve.

mtpy.modeling.occam1d.plot_response module

Created on Mon Oct 30 13:33:37 2023

@author: jpeacock

```
class mtpy.modeling.occam1d.plot_response.Plot1DResponse(data_te_fn=None, data_tm_fn=None,
                                                          model_fn=None, resp_te_fn=None,
                                                          resp_tm_fn=None, iter_te_fn=None,
                                                          iter_tm_fn=None, **kwargs)
```

Bases: object

Plot the 1D response and model.

Plots apparent resistivity and phase

in different subplots with the model on the far right. You can plot both

TE and TM modes together along with different iterations of the model. These will be plotted in different colors or shades of gray depending on color_scale.

Example

```
>>> import mtpy.modeling.occam1d as occam1d
>>> p1 = occam1d.Plot1DResponse(plot_yn='n')
>>> p1.data_te_fn = r"/home/occam1d/mt01/TE/Occam_DataFile_TE.dat"
>>> p1.data_tm_fn = r"/home/occam1d/mt01/TM/Occam_DataFile_TM.dat"
>>> p1.model_fn = r"/home/occam1d/mt01/TE/Model1D"
>>> p1.iter_te_fn = [r"/home/occam1d/mt01/TE/TE_{0}.iter".
    >>>                   format(ii)
    >>> ...
    >>>             for ii in range(5,10)]
    >>> p1.iter_tm_fn = [r"/home/occam1d/mt01/TM/TM_{0}.iter".
    >>>                   format(ii)
    >>> ...
    >>>             for ii in range(5,10)]
    >>> p1.resp_te_fn = [r"/home/occam1d/mt01/TE/TE_{0}.resp".
    >>>                   format(ii)
    >>> ...
    >>>             for ii in range(5,10)]
    >>> p1.resp_tm_fn = [r"/home/occam1d/mt01/TM/TM_{0}.resp".
    >>>                   format(ii)
    >>> ...
    >>>             for ii in range(5,10)]
    >>> p1.plot()
```

Attributes	Description
axm	matplotlib.axes instance for model subplot
axp	matplotlib.axes instance for phase subplot
axr	matplotlib.axes instance for app. res subplot
color_mode	[‘color’ ‘bw’]
cted	color of TE data markers
ctem	color of TM data markers
ctmd	color of TE model markers
ctmm	color of TM model markers
data_te_fn	full path to data file for TE mode
data_tm_fn	full path to data file for TM mode
depth_limits	(min, max) limits for depth plot in depth_units
depth_scale	[‘log’ ‘linear’] default is linear

continues on next page

Table 3 – continued from previous page

Attributes	Description
depth_units	[‘m’ ‘km’] *default is ‘km’
e_capsize	capsize of error bars
e_caphick	cap thickness of error bars
fig	matplotlib.figure instance for plot
fig_dpi	resolution in dots-per-inch for figure
fig_num	number of figure instance
fig_size	size of figure in inches [width, height]
font_size	size of axes tick labels, axes labels are +2
grid_alpha	transparency of grid
grid_color	color of grid
iter_te_fn	full path or list of .iter files for TE mode
iter_tm_fn	full path or list of .iter files for TM mode
lw	width of lines for model
model_fn	full path to model file
ms	marker size
mtd	marker for TE data
mtem	marker for TM data
mtmd	marker for TE model
mtmm	marker for TM model
phase_limits	(min, max) limits on phase in degrees
phase_major_ticks	spacing for major ticks in phase
phase_minor_ticks	spacing for minor ticks in phase
plot_yn	[‘y’ ‘n’] plot on instantiation
res_limits	limits of resistivity in linear scale
resp_te_fn	full path or list of .resp files for TE mode
resp_tm_fn	full path or list of .iter files for TM mode
subplot_bottom	spacing of subplots from bottom of figure
subplot_hspace	height spacing between subplots
subplot_left	spacing of subplots from left of figure
subplot_right	spacing of subplots from right of figure
subplot_top	spacing of subplots from top of figure
subplot_wspace	width spacing between subplots
title_str	title of plot

plot()

Plot data, response and model.

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.plotAllResponses()
>>> #change line width
>>> p1.lw = 2
>>> 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.

Arguments:

```
**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(fig)

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.plotAllResponses()
>>> [ax.grid(True, which='major') for ax in [ps1.axrte,ps1.axtep]]
>>> ps1.update_plot()
```

mtpy.modeling.occam1d.run module

Created on Mon Oct 30 13:35:16 2023

@author: jpeacock

```
class mtpy.modeling.occam1d.run.Occam1DRun(startup_fn=None, occam_path=None, **kwargs)
```

Bases: object

Run occam 1d from python given the correct files and location of occam1d executable

```
run_occam1d()
```

Run occam1d.

mtpy.modeling.occam1d.startup module

Created on Mon Oct 30 13:32:42 2023

@author: jpeacock

```
class mtpy.modeling.occam1d.startup.Occam1DStartup(data_fn=None, model_fn=None, **kwargs)
```

Bases: object

read and write input files for Occam1D

Attributes	Description
_ss	string spacing
_startup_fn	basename of startup file <i>default</i> is OccamStartup1D
data_fn	full path to data file
debug_level	debug level <i>default</i> is 1
description	description of inversion for your self <i>default</i> is 1D_Occam_Inv
max_iter	maximum number of iterations <i>default</i> is 20
model_fn	full path to model file
rough_type	roughness type <i>default</i> is 1
save_path	full path to save files to
start_iter	first iteration number <i>default</i> is 0
start_lagrange	starting lagrange number on log scale <i>default</i> is 5
start_misfit	starting misfit value <i>default</i> is 100
start_rho	starting resistivity value (halfspace) in log scale <i>default</i> is 100
start_rough	starting roughness (ignored by Occam1D) <i>default</i> is 1E7
startup_fn	full path to startup file
target_rms	target rms <i>default</i> is 1.0

```
property data_fn  
property model_fn  
read_startup_file(startup_fn)  
    reads in a 1D input file
```

Arguments:

inputfn : full path to input file

Returns:

Occam1D.indict : dictionary with keys following the header and
‘res’ : an array of resistivity values

Example

```
>>> old = occam.Occam1D()  
>>> old.savepath = r"/home/Occam1D/Line1/Inv1_TE"  
>>> old.read1DInputFile()
```

write_startup_file(*save_path=None*, ***kwargs*)

Make a 1D input file for Occam 1D

Arguments:**savepath**

[full path to save input file to, if just path then] saved as savepath/input

model_fn

[full path to model file, if None then assumed to be in] savepath/model.mod

data_fn

[full path to data file, if None then assumed to be] in savepath/TE.dat or TM.dat

rough_type : roughness type. *default* = 0**max_iter** : maximum number of iterations. *default* = 20**target_rms** : target rms value. *default* = 1.0**start_rho**

[starting resistivity value on linear scale.] *default* = 100

description : description of the inversion.**start_lagrange**

[starting Lagrange multiplier for smoothness.] *default* = 5

start_rough : starting roughness value. *default* = 1E7**debuglevel**

[something to do with how Fortran debuggs the code] Almost always leave at *default* = 1

start_iter

[the starting iteration number, handy if the] starting model is from a previous run. *default* = 0

start_misfit : starting misfit value. *default* = 100**Returns:**

Occam1D.inputfn : full path to input file.

Example

```

>>> old = occam.Occam1D()
>>> old.make1DdataFile('MT01',edipath=r"/home/Line1",
>>>                               savepath=r"/home/Occam1D/Line1/Inv1_TE",
>>>                               mode='TE')
>>> Wrote Data File: /home/Occam1D/Line1/Inv1_TE/MT01TE.dat
>>>
>>> old.make1DModelFile(savepath=r"/home/Occam1D/Line1/Inv1_TE",
>>>                               nlayers=50, bottomlayer=1000, z1layer=50)
>>> Wrote Model file: /home/Occam1D/Line1/Inv1_TE/Model1D
>>>
>>> old.make1DInputFile(rhostart=10, targetrms=1.5, maxiter=15)
>>> Wrote Input File: /home/Occam1D/Line1/Inv1_TE/Input1D

```

mtpy.modeling.occam1d.tools module

Module contents

Created on Mon Oct 30 13:56:36 2023

@author: jpeacock

class mtpy.modeling.occam1d.Occam1DData(*mt_dataframe*, *kwargs*)**

Bases: `object`

reads and writes occam 1D data files

Attributes	Description
<code>_data_fn</code>	basename of data file <i>default</i> is Occam1DDataFile
<code>_header_line</code>	header line for description of data columns
<code>_ss</code>	string spacing <i>default</i> is 6*' '
<code>_string_fmt</code>	format of data <i>default</i> is '+.6e'
<code>data</code>	array of data
<code>data_fn</code>	full path to data file
<code>freq</code>	frequency array of data
<code>mode</code>	mode to invert for ['TE' 'TM' 'det']
<code>phase_te</code>	array of TE phase
<code>phase_tm</code>	array of TM phase
<code>res_te</code>	array of TE apparent resistivity
<code>res_tm</code>	array of TM apparent resistivity
<code>resp_fn</code>	full path to response file
<code>save_path</code>	path to save files to

Methods	Description
<code>write_data_file</code>	write an Occam1D data file
<code>read_data_file</code>	read an Occam1D data file
<code>read_resp_file</code>	read a .resp file output by Occam1D

Example

```
>>> import mtpy.modeling.occam1d as occam1d
>>> #--> make a data file for TE mode
>>> d1 = occam1d.Data()
>>> d1.write_data_file(edi_file=r'/home/MT/mt01.edi', res_err=10,
>>>     phase_err=2.5,
>>>     ...                               save_path=r"/home/occam1d/mt01/TE", mode='TE')
```

property mode

property mode_01

property mode_02

read_data_file(data_fn)

reads a 1D data file

Arguments:

data_fn : full path to data file

Returns:

Occam1D.rpdct : dictionary with keys:

'freq' : an array of frequencies with length nf

'resxy'
[TE resistivity array with shape (nf,4) for (0) data,]
(1) dataerr, (2) model, (3) modelerr

'resyx'
[TM resistivity array with shape (nf,4) for (0) data,]
(1) dataerr, (2) model, (3) modelerr

'phasexy'
[TE phase array with shape (nf,4) for (0) data,]
(1) dataerr, (2) model, (3) modelerr

'phaseyx'
[TM phase array with shape (nf,4) for (0) data,]
(1) dataerr, (2) model, (3) modelerr

Example

```
>>> old = occam1d.Data()
>>> old.data_fn = r"/home/Occam1D/Line1/Inv1_TE/MT01TE.dat"
>>> old.read_data_file()
```

read_resp_file(resp_fn=None, data_fn=None)

read response file

resp_fn : full path to response file

data_fn : full path to data file

freq : an array of frequencies with length nf

res_te
 [TE resistivity array with shape (nf,4) for (0) data,]
 (1) dataerr, (2) model, (3) modelerr

res_tm
 [TM resistivity array with shape (nf,4) for (0) data,]
 (1) dataerr, (2) model, (3) modelerr

phase_te
 [TE phase array with shape (nf,4) for (0) data,]
 (1) dataerr, (2) model, (3) modelerr

phase_tm
 [TM phase array with shape (nf,4) for (0) data,]
 (1) dataerr, (2) model, (3) modelerr

Example

```
::      >>> o1d      =      occam1d.Data()      >>>      o1d.data_fn      =
r"/home/occam1d/mt01/TE/Occam1D_DataFile_TE.dat"      >>>
o1d.read_resp_file(r"/home/occam1d/mt01/TE/TE_7.resp")
```

write_data_file(filename, mode='det', remove_outofquadrant=False)

make1Ddatafile will write a data file for Occam1D

Arguments:

rp_tuple
 [np.ndarray (freq, res, res_err, phase, phase_err)] with res, phase having shape (num_freq, 2, 2).

edi_file
 [string] full path to edi file to be modeled.

save_path
 [string] path to save the file, if None set to dirname of station if edipath = None. Otherwise set to dirname of edipath.

thetar
 [float] rotation angle to rotate Z. Clockwise positive and N=0 *default* = 0

mode
 [[‘te’ | ‘tm’ | ‘det’]]

mode to model can be (*default*='both'):

- ‘te’ for just TE mode (res/phase)
- ‘tm’ for just TM mode (res/phase)
- **‘det’ for the determinant of Z (converted to res/phase)**

add ‘z’ to any of these options to model impedance tensor values instead of res/phase

res_err
 [float] errorbar for resistivity values. Can be set to (*default* = ‘data’):

- ‘data’ for errorbars from the data
- percent number ex. 10 for ten percent

phase_err

[float] errorbar for phase values. Can be set to (*default* = ‘data’):

- ‘data’ for errorbars from the data
- percent number ex. 10 for ten percent

res_errorfloor: float

error floor for resistivity values in percent

phase_errorfloor: float

error floor for phase in degrees

remove_outofquadrant: True/False; option to remove the resistivity and

phase values for points with phases out of the 1st/3rd quadrant (occam requires $0 < \text{phase} < 90$ degrees; phases in the 3rd quadrant are shifted to the first by adding 180 degrees)

Example

```
>>> import mtpy.modeling.occam1d as occam1d
>>> #--> make a data file
>>> d1 = occam1d.Data()
>>> d1.write_data_file(edi_file=r'/home/MT/mt01.edi', res_err=10,
>>> ...                      phase_err=2.5, mode='TE',
>>> ...                      save_path=r"/home/occam1d/mt01/TE")
```

class mtpy.modeling.occam1d.Occam1DModel(*model_fn=None*, ***kwargs*)

Bases: object

read and write the model file fo Occam1D

All depth measurements are in meters.

Attributes	Description
_model_fn	basename for model file <i>default</i> is Model1D
_ss	string spacing in model file <i>default</i> is 3*‘ ’
_string_fmt	format of model layers <i>default</i> is ‘.0f’
air_layer_height	height of air layer <i>default</i> is 10000
bottom_layer	bottom of the model <i>default</i> is 50000
itdict	dictionary of values from iteration file
iter_fn	full path to iteration file
model_depth	array of model depths
model_fn	full path to model file
model_penalty	array of penalties for each model layer
model_preference_penalty	array of model preference penalties for each layer
model_preference	array of preferences for each layer
model_res	array of resistivities for each layer
n_layers	number of layers in the model
num_params	number of parameters to invert for (n_layers+2)
pad_z	padding of model at depth <i>default</i> is 5 blocks
save_path	path to save files
target_depth	depth of target to investigate
z1_layer	depth of first layer <i>default</i> is 10

Methods	Description
write_model_file	write an Occam1D model file, where depth increases on a logarithmic scale
read_model_file	read an Occam1D model file
read_iter_file	read an .iter file output by Occam1D

Example

```
>>> #--> make a model file
>>> m1 = occam1d.Model()
>>> m1.write_model_file(save_path=r"/home/occam1d/mt01/TE")
```

read_iter_file(iter_fn=None, model_fn=None)

read an 1D iteration file

Arguments:

imode : mode to read from

Returns:

Occam1D.itdict : dictionary with keys of the header:

model_res
[fills this array with the appropriate] values (0) for data, (1) for model

Example

```
>>> m1 = occam1d.Model()
>>> m1.model_fn = r"/home/occam1d/mt01/TE/Model1D"
>>> m1.read_iter_file(r"/home/Occam1D/Inv1_TE/M01TE_15.iter")
```

read_model_file(model_fn=None)

will read in model 1D file

Arguments:

modelfn : full path to model file

Fills attributes:

- **model_depth'** : depth of model in meters
- **model_res** : value of resistivity
- **model_penalty** : penalty
- **model_preference** : preference
- **model_penalty_preference** : preference penalty

Example

```
>>> m1 = occam1d.Model()
>>> m1.savepath = r"/home/Occam1D/Line1/Inv1_TE"
>>> m1.read_model_file()
```

write_model_file(*save_path=None*, *kwargs*)**

Makes a 1D model file for Occam1D.

Arguments:

save_path : path to save file to, if just path saved as
savepathmodel.mod, if None defaults to dirpath

n_layers : number of layers

bottom_layer : depth of bottom layer in meters

target_depth : depth to target under investigation

pad_z : padding on bottom of model past target_depth

z1_layer : depth of first layer in meters

air_layer_height : height of air layers in meters

Returns:

Occam1D.modelfn = full path to model file

..Note: This needs to be redone.

Example

```
>>> old = occam.Occam1D()
>>> old.make1DModelFile(savepath=r"/home/Occam1D/Line1/Inv1_TE",
>>>                               nlayers=50, bottomlayer=10000, z1layer=50)
>>> Wrote Model file: /home/Occam1D/Line1/Inv1_TE/Model1D
```

class mtpy.modeling.occam1d.Occam1DRun(*startup_fn=None*, *occam_path=None*, *kwargs*)**

Bases: object

Run occam 1d from python given the correct files and location of occam1d executable

run_occam1d()

Run occam1d.

class mtpy.modeling.occam1d.Occam1DStartup(*data_fn=None*, *model_fn=None*, *kwargs*)**

Bases: object

read and write input files for Occam1D

Attributes	Description
_ss	string spacing
_startup_fn	basename of startup file <i>default</i> is OccamStartup1D
data_fn	full path to data file
debug_level	debug level <i>default</i> is 1
description	description of inversion for your self <i>default</i> is 1D_Occam_Inv
max_iter	maximum number of iterations <i>default</i> is 20
model_fn	full path to model file
rough_type	roughness type <i>default</i> is 1
save_path	full path to save files to
start_iter	first iteration number <i>default</i> is 0
start_lagrange	starting lagrange number on log scale <i>default</i> is 5
start_misfit	starting misfit value <i>default</i> is 100
start_rho	starting resistivity value (halfspace) in log scale <i>default</i> is 100
start_rough	starting roughness (ignored by Occam1D) <i>default</i> is 1E7
startup_fn	full path to startup file
target_rms	target rms <i>default</i> is 1.0

property data_fn
property model_fn
read_startup_file(*startup_fn*)
 reads in a 1D input file

Arguments:

inputfn : full path to input file

Returns:

Occam1D.indict : dictionary with keys following the header and
 ‘res’ : an array of resistivity values

Example

```
>>> old = occam.Occam1D()  

>>> old.savepath = r"/home/Occam1D/Line1/Inv1_TE"  

>>> old.read1DInputFile()
```

write_startup_file(*save_path=None*, **kwargs)

Make a 1D input file for Occam 1D

Arguments:

savepath
 [full path to save input file to, if just path then] saved as savepath/input
model_fn
 [full path to model file, if None then assumed to be in] savepath/model.mod

data_fn
[full path to data file, if None then assumed to be] in savepath/TE.dat or TM.dat

rough_type : roughness type. *default* = 0

max_iter : maximum number of iterations. *default* = 20

target_rms : target rms value. *default* = 1.0

start_rho
[starting resistivity value on linear scale.] *default* = 100

description : description of the inversion.

start_lagrange
[starting Lagrange multiplier for smoothness.] *default* = 5

start_rough : starting roughness value. *default* = 1E7

debuglevel
[something to do with how Fortran debuggs the code] Almost always leave at *default* = 1

start_iter
[the starting iteration number, handy if the] starting model is from a previous run. *default* = 0

start_misfit : starting misfit value. *default* = 100

Returns:

Occam1D.inputfn : full path to input file.

Example

```
>>> old = occam.Occam1D()
>>> old.make1DdataFile('MT01',edipath=r"/home/Line1",
>>>                               savepath=r"/home/Occam1D/Line1/Inv1_TE",
>>>                               mode='TE')
>>> Wrote Data File: /home/Occam1D/Line1/Inv1_TE/MT01TE.dat
>>>
>>> old.make1DModelFile(savepath=r"/home/Occam1D/Line1/Inv1_TE",
>>>                               nlayers=50,bottomlayer=10000,z1layer=50)
>>> Wrote Model file: /home/Occam1D/Line1/Inv1_TE/Model1D
>>>
>>> old.make1DInputFile(rhostart=10,targetrms=1.5,maxiter=15)
>>> Wrote Input File: /home/Occam1D/Line1/Inv1_TE/Input1D
```

```
class mtpy.modeling.occam1d.Plot1DResponse(data_te_fn=None, data_tm_fn=None, model_fn=None,
                                             resp_te_fn=None, resp_tm_fn=None, iter_te_fn=None,
                                             iter_tm_fn=None, **kwargs)
```

Bases: object

Plot the 1D response and model.

Plots apparent resistivity and phase

in different subplots with the model on the far right. You can plot both

TE and TM modes together along with different iterations of the model. These will be plotted in different colors or shades of gray depneng on color_scale.

Example

```
>>> import mtpy.modeling.occam1d as occam1d
>>> p1 = occam1d.Plot1DResponse(plot_yn='n')
>>> p1.data_te_fn = r"/home/occam1d/mt01/TE/Occam_DataFile_TE.dat"
>>> p1.data_tm_fn = r"/home/occam1d/mt01/TM/Occam_DataFile_TM.dat"
>>> p1.model_fn = r"/home/occam1d/mt01/TE/Model1D"
>>> p1.iter_te_fn = [r"/home/occam1d/mt01/TE/TE_{0}.iter".
    ~format(ii)
>>> ...           for ii in range(5,10)]
>>> p1.iter_tm_fn = [r"/home/occam1d/mt01/TM/TM_{0}.iter".
    ~format(ii)
>>> ...           for ii in range(5,10)]
>>> p1.resp_te_fn = [r"/home/occam1d/mt01/TE/TE_{0}.resp".
    ~format(ii)
>>> ...           for ii in range(5,10)]
>>> p1.resp_tm_fn = [r"/home/occam1d/mt01/TM/TM_{0}.resp".
    ~format(ii)
>>> ...           for ii in range(5,10)]
>>> p1.plot()
```

Attributes	Description
axm	matplotlib.axes instance for model subplot
axp	matplotlib.axes instance for phase subplot
axr	matplotlib.axes instance for app. res subplot
color_mode	[‘color’ ‘bw’]
cted	color of TE data markers
ctem	color of TM data markers
ctmd	color of TE model markers
ctmm	color of TM model markers
data_te_fn	full path to data file for TE mode
data_tm_fn	full path to data file for TM mode
depth_limits	(min, max) limits for depth plot in depth_units
depth_scale	[‘log’ ‘linear’] <i>default</i> is linear
depth_units	[‘m’ ‘km’] * <i>default</i> is ‘km’
e_capsize	capsize of error bars
e_capthick	cap thickness of error bars
fig	matplotlib.figure instance for plot
fig_dpi	resolution in dots-per-inch for figure
fig_num	number of figure instance
fig_size	size of figure in inches [width, height]
font_size	size of axes tick labels, axes labels are +2
grid_alpha	transparency of grid
grid_color	color of grid
iter_te_fn	full path or list of .iter files for TE mode
iter_tm_fn	full path or list of .iter files for TM mode
lw	width of lines for model
model_fn	full path to model file
ms	marker size
mted	marker for TE data
mtem	marker for TM data
mtmd	marker for TE model

continues on next page

Table 4 – continued from previous page

Attributes	Description
mtmm	marker for TM model
phase_limits	(min, max) limits on phase in degrees
phase_major_ticks	spacing for major ticks in phase
phase_minor_ticks	spacing for minor ticks in phase
plot_yn	[‘y’ ‘n’] plot on instantiation
res_limits	limits of resistivity in linear scale
resp_te_fn	full path or list of .resp files for TE mode
resp_tm_fn	full path or list of .iter files for TM mode
subplot_bottom	spacing of subplots from bottom of figure
subplot_hspace	height spacing between subplots
subplot_left	spacing of subplots from left of figure
subplot_right	spacing of subplots from right of figure
subplot_top	spacing of subplots from top of figure
subplot_wspace	width spacing between subplots
title_str	title of plot

plot()

Plot data, response and model.

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.Occam2DDData(r"/home/occam2d/Data.dat")
>>> p1 = ocd.plotAllResponses()
>>> #change line width
>>> p1.lw = 2
>>> 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.

Arguments:

```
**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...
```

(continues on next page)

(continued from previous page)

```

**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.Occam2DDData(dfn)
>>> ps1 = ocd.plotPseudoSection()
>>> ps1.save_plot(r'/home/MT/figures', file_format='jpg')

```

update_plot(fig)

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.Occam2DDData(dfn)
>>> ps1 = ocd.plotAllResponses()
>>> [ax.grid(True, which='major') for ax in [ps1.axrte,ps1.axtep]]
>>> ps1.update_plot()

```

class mtpy.modeling.occam1d.PlotOccam1DL2(dir_path, model_fn, **kwargs)

Bases: *PlotBase*

Plot L2 curve of iteration vs rms and roughness.

Arguments::

rms_arr
[structured array with keys:]

- ‘iteration’ –> for iteration number (int)
- ‘rms’ –> for rms (float)
- ‘roughness’ –> for roughness (float)

Keywords/attributes	Description
ax1	matplotlib.axes instance for rms vs iteration
ax2	matplotlib.axes instance for roughness vs rms
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 axes tick labels, axes labels is +2
plot_yn	[‘y’ ‘n’] ‘y’ –> to plot on instantiation ‘n’ –> to not plot on instantiation
rms_arr	structure np.array as described above
rms_color	color of rms marker and line
rms_lw	line width of rms line
rms_marker	marker for rms values
rms_marker_size	size of marker for rms values
rms_mean_color	color of mean line
rms_median_color	color of median line
rough_color	color of roughness line and marker
rough_font_size	font size for iteration number inside roughness marker
rough_lw	line width for roughness line
rough_marker	marker for roughness
rough_marker_size	size of marker for roughness
subplot_bottom	subplot spacing from bottom
subplot_left	subplot spacing from left
subplot_right	subplot spacing from right
subplot_top	subplot spacing from top

plot()

Plot L2 curve.

mtpy.modeling.occam2d package**Submodules****mtpy.modeling.occam2d.data module**

Created on Tue Mar 7 19:01:14 2023

@author: jpeacock

class mtpy.modeling.occam2d.data.Occam2DData(dataframe=None, center_point=None, **kwargs)

Bases: object

Reads and writes data files and more.

Inherits Profile, so the intended use is to use Data to project stations onto a profile, then write the data file.

Model Modes	Description
1 or log_all	Log resistivity of TE and TM plus Tipper
2 or log_te_tip	Log resistivity of TE plus Tipper
3 or log_tm_tip	Log resistivity of TM plus Tipper
4 or log_te_tm	Log resistivity of TE and TM
5 or log_te	Log resistivity of TE
6 or log_tm	Log resistivity of TM
7 or all	TE, TM and Tipper
8 or te_tip	TE plus Tipper
9 or tm_tip	TM plus Tipper
10 or te_tm	TE and TM mode
11 or te	TE mode
12 or tm	TM mode
13 or tip	Only Tipper

Example Write Data File

```
>>> from mtpy.modeling.occam2d import Data
>>> occam_data_object = Data()
>>> occam_data_object.read_data_file(r"path/to/data/file.dat")
>>> occam_data_object.model_mode = 2
>>> occam_data_object.write_data_file(r"path/to/new/data/file_te.dat")
```

property data_filename

Data filename.

property dataframe

Dataframe function.

property frequencies

Frequencies function.

mask_from_datafile(mask_datafn)

Reads a separate data file and applies mask from this data file.

mask_datafn needs to have exactly the same frequencies, and station names must match exactly.

property n_data

N data.

property n_frequencies

N frequencies.

property n_stations

N stations.

property offsets

Offsets function.

read_data_file(data_fn=None)

Read in an existing data file and populate appropriate attributes

- data
- data_list

- freq
- station_list
- station_locations

Arguments::**data_fn**

[string] full path to data file *default* is None and set to save_path/fn_basename

Example

```
>>> import mtpy.modeling.occam2d as occam2d
>>> ocd = occam2d.Data()
>>> ocd.read_data_file(r"/home/Occam2D/Line1/Inv1/Data.dat")
```

property stations

Stations function.

write_data_file(data_fn=None)

Write a data file.

Arguments::**data_fn**

[string] full path to data file. *default* is save_path/fn_basename

If there data is None, then _fill_data is called to create a profile, rotate data and get all the necessary data. This way you can use write_data_file directly without going through the steps of projecting the stations, etc.

Example

```
:: >>> edipath = r"/home/mt/edi_files" >>> slst = ['mt{:0:03}'.format(ss) for ss in range(1, 20)] >>> ocd = occam2d.Data(edi_path=edipath, station_list=slst) >>> ocd.save_path = r"/home/occam/line1/inv1" >>> ocd.write_data_file()
```

mtpy.modeling.occam2d.mesh module

Created on Tue Mar 7 18:02:57 2023

@author: jpeacock

class mtpy.modeling.occam2d.mesh.Mesh(station_locations=None, **kwargs)

Bases: object

deals only with the finite element mesh. Builds a finite element mesh based on given parameters defined below. The mesh reads in the station locations, finds the center and makes the relative location of the furthest left hand station 0. The mesh increases in depth logarithmically as required by the physics of MT. Also, the model extends horizontally and vertically with padding cells in order to fulfill the assumption of the forward operator that at the edges the structure is 1D. Stations are placed on the horizontal nodes as required by Wannamaker's forward operator.

Mesh has the ability to create a mesh that incorporates topography given a elevation profile. It adds more cells to the mesh with thickness z1_layer. It then sets the values of the triangular elements according to the elevation value at that location. If the elevation covers less than 50% of the triangular cell, then the cell value is set to that of air

 **Note**

Mesh is inherited by Regularization, so the mesh can also be built from there, same as the example below.

Arguments:

Key Words/Attrib	Description
air_key	letter associated with the value of air <i>default</i> is 0
air_value	value given to an air cell, <i>default</i> is 1E13
cell_width	width of cells within station area in meters <i>default</i> is 100
elevation_profile	elevation profile along the profile line. given as np.ndarray(nx, 2), where the elements are x_location, elevation. If elevation profile is given add_elevation is called automatically. <i>default</i> is None
mesh_fn	full path to mesh file.
mesh_values	letter values of each triangular mesh element if the cell is free value is ?
n_layers	number of vertical layers in mesh <i>default</i> is 90
num_x_pad_	number of horizontal padding cells outside the station area that will increase in size by x_pad_multiplier. <i>default</i> is 7
num_x_pad_	number of horizontal padding cells just outside the station area with width cell_width. This is to extend the station area if needed. <i>default</i> is 2
num_z_pad_	number of vertical padding cells below z_target_depth down to z_bottom. <i>default</i> is 5
rel_station_lc	relative station locations within the mesh. The locations are relative to the center of the station area. <i>default</i> is None, filled later
save_path	full path to save mesh file to. <i>default</i> is current working directory.
station_location	location of stations in meters, can be on a relative grid or in UTM.
x_grid	location of horizontal grid nodes in meters
x_nodes	relative spacing between grid nodes
x_pad_multip	horizontal padding cells will increase by this multiple out to the edge of the grid. <i>default</i> is 1.5
z1_layer	thickness of the first layer in the model. Should be at least 1/4 of the first skin depth <i>default</i> is 10
z_bottom	bottom depth of the model (m). Needs to be large enough to be 1D at the edge. <i>default</i> is 200000.0
z_grid	location of vertical nodes in meters
z_nodes	relative distance between vertical nodes in meters
z_target_dept	depth to deepest target of interest. Below this depth cells will be padded to z_bottom

Methods	Description
add_elevation	adds elevation to the mesh given elevation profile.
build_mesh	builds the mesh given the attributes of Mesh. If elevation_profile is not None, add_elevation is called inside build_mesh
plot_mesh	plots the built mesh with station location.
read_mesh_file	reads in an existing mesh file and populates the appropriate attributes.
write_mesh_file	writes a mesh file to save_path

Example

```
>>> import mtpy.modeling.occam2d as occcam2d
>>> edipath = r"/home/mt/edi_files"
>>> slist = ['mt{:03}'.format(ss) for ss in range(20)]
>>> ocd = occcam2d.Data(edi_path=edipath, station_list=slist)
>>> ocd.save_path = r"/home/occam/Line1/Inv1"
>>> ocd.write_data_file()
>>> ocm = occcam2d.Mesh(ocd.station_locations)
>>> # add in elevation
>>> ocm.elevation_profile = ocd.elevation_profile
>>> # change number of layers
>>> ocm.n_layers = 110
>>> # change cell width in station area
>>> ocm.cell_width = 200
>>> ocm.build_mesh()
>>> ocm.plot_mesh()
>>> ocm.save_path = ocd.save_path
>>> ocm.write_mesh_file()
```

add_elevation(elevation_profile=None)

the elevation model needs to be in relative coordinates and be a numpy.ndarray(2, num_elevation_points) where the first column is the horizontal location and the second column is the elevation at that location.

If you have a elevation model use Profile to project the elevation information onto the profile line

To build the elevation I'm going to add the elevation to the top of the model which will add cells to the mesh. there might be a better way to do this, but this is the first attempt. So I'm going to assume that the first layer of the mesh without elevation is the minimum elevation and blocks will be added to max elevation at an increment according to z1_layer

Note

the elevation model should be symmetrical ie, starting at the first station and ending on the last station, so for now any elevation outside the station area will be ignored and set to the elevation of the station at the extremities. This is not ideal but works for now.

Arguments:**elevation_profile**

[np.ndarray(2, num_elev_points)]

- 1st row is for profile location
- 2nd row is for elevation values

Computes:**mesh_values**

[mesh values, setting anything above topography] to the key for air, which for Occam is '0'

build_mesh()

Build the finite element mesh given the parameters defined by the attributes of Mesh. Computes relative station locations by finding the center of the station area and setting the middle to 0. Mesh blocks are built by calculating the distance between stations and putting evenly spaced blocks between the stations being

close to cell_width. This places a horizontal node at the station location. If the spacing between stations is smaller than cell_width, a horizontal node is placed between the stations to be sure the model has room to change between the station.

If elevation_profile is given, add_elevation is called to add topography into the mesh.

Populates attributes:

- mesh_values
- rel_station_locations
- x_grid
- x_nodes
- z_grid
- z_nodes

Example

```
::      >>> import mtpy.modeling.occam2d as occcam2d >>> edipath = r"/home/mt/edi_files" >>> slist = ['mt{:03}'.format(ss) for ss in range(20)] >>> ocd = occcam2d.Data(edipath=edipath, station_list=slist) >>> ocd.save_path = r"/home/occam/Line1/Inv1" >>> ocd.write_data_file() >>> ocm = occcam2d.Mesh(ocd.station_locations) >>> # add in elevation >>> ocm.elevation_profile = ocd.elevation_profile >>> # change number of layers >>> ocm.n_layers = 110 >>> # change cell width in station area >>> ocm.cell_width = 200 >>> ocm.build_mesh()
```

`plot_mesh(**kwargs)`

Plot built mesh with station locations.

Key Words	Description
depth_scale	[‘km’ ‘m’] scale of mesh plot. <i>default</i> is ‘km’
fig_dpi	dots-per-inch resolution of the figure <i>default</i> is 300
fig_num	number of the figure instance <i>default</i> is ‘Mesh’
fig_size	size of figure in inches (width, height) <i>default</i> is [5, 5]
fs	size of font of axis tick labels, axis labels are fs+2. <i>default</i> is 6
ls	[‘-’ ‘.’ ‘:’] line style of mesh lines <i>default</i> is ‘-’
marker	marker of stations <i>default</i> is r'\$\blacktriangleleft\$'
ms	size of marker in points. <i>default</i> is 5
plot_triangles	[‘y’ ‘n’] to plot mesh triangles. <i>default</i> is ‘n’

`read_mesh_file(mesh_fn)`

reads an occam2d 2D mesh file

Arguments:

mesh_fn
[string] full path to mesh file

Populates:

x_grid : array of horizontal locations of nodes (m)

x_nodes: array of horizontal node relative distances
(column locations (m))

z_grid : array of vertical node locations (m)

z_nodes

[array of vertical nodes] (row locations(m))

mesh_values : np.array of free parameters

To do:

incorporate fixed values

Example

```
>>> import mtpy.modeling.occam2d as occam2d
>>> mg = occam2d.Mesh()
>>> mg.mesh_fn = r"/home/mt/occam/line1/Occam2Dmesh"
>>> mg.read_mesh_file()
```

write_mesh_file(*save_path=None, basename='Occam2DMesh'*)

Write a finite element mesh file.

Calls build_mesh if it already has not been called.

Arguments:

save_path

[string] directory path or full path to save file

basename

[string] basename of mesh file. *default* is ‘Occam2DMesh’

Returns:

mesh_fn

[string] full path to mesh file

example

```
>>> import mtpy.modeling.occam2d as occam2d
>>> edi_path = r"/home/mt/edi_files"
>>> profile = occam2d.Profile(edi_path)
>>> profile.plot_profile()
>>> mesh = occam2d.Mesh(profile.station_locations)
>>> mesh.build_mesh()
>>> mesh.write_mesh_file(save_path=r"/home/occam2d/Inv1")
```

mtpy.modeling.occam2d.model module

Created on Tue Mar 7 19:11:57 2023

@author: jpeacock

class mtpy.modeling.occam2d.model.Occam2DModel(*iter_fn=None, model_fn=None, mesh_fn=None, **kwargs*)

Bases: *Startup*

Read .iter file output by Occam2d. Builds the resistivity model from mesh and regularization files found from the .iter file. The resistivity model is an array(x_nodes, z_nodes) set on a regular grid, and the values of the model response are filled in according to the regularization grid. This allows for faster plotting.

Inherits Startup because they are basically the same object.

Argument:

iter_fn

[string] full path to .iter file to read. *default* is None.

model_fn

[string] full path to regularization file. *default* is None and found directly from the .iter file.
Only input if the regularization is different from the file that is in the .iter file.

mesh_fn

[string] full path to mesh file. *default* is None Found directly from the model_fn file. Only input if the mesh is different from the file that is in the model file.

Key Words/Attributes	Description
data_fn	full path to data file
iter_fn	full path to .iter file
mesh_fn	full path to mesh file
mesh_x	np.ndarray(x_nodes, z_nodes) mesh grid for plotting
mesh_z	np.ndarray(x_nodes, z_nodes) mesh grid for plotting
model_values	model values from startup file
plot_x	nodes of mesh in horizontal direction
plot_z	nodes of mesh in vertical direction
res_model	np.ndarray(x_nodes, z_nodes) resistivity model values in linear scale

Methods	Description
build_model	get the resistivity model from the .iter file in a regular grid according to the mesh file with resistivity values according to the model file
read_iter_file	read .iter file and fill appropriate attributes
write_iter_file	write an .iter file incase you want to set it as the starting model or a priori model

Example

```
::      >>> model = occam2D.Model(r"/home/occam/line1/inv1/test_01.iter")    >>>
model.build_model()
```

build_model()

build the model from the mesh, regularization grid and model file

read_iter_file(iter_fn=None)

Read an iteration file.

Arguments:

iter_fn

[string] full path to iteration file if iterpath=None. If iterpath is input then iterfn is just the name of the file without the full path.

Returns:**Example**

```
>>> import mtpy.modeling.occam2d as occam2d
>>> itfn = r"/home/0Occam2D/Line1/Inv1/Test_15.iter"
>>> ocm = occam2d.Model(itfn)
>>> ocm.read_iter_file()
```

write_iter_file(iter_fn=None)

write an iteration file if you need to for some reason, same as startup file

mtpy.modeling.occam2d.regularization module

Created on Tue Mar 7 18:13:52 2023

@author: jpeacock

class mtpy.modeling.occam2d.regularization.Regularization(station_locations=None, **kwargs)

Bases: *Mesh*

Creates a regularization grid based on Mesh.

Note that Mesh is inherited

by Regularization, therefore the intended use is to build a mesh with

the Regularization class.

The regularization grid is what Occam calculates the inverse model on. Setup is tricky and can be painful, as you can see it is not quite fully functional yet, as it cannot incorporate topography yet. It seems like you'd like to have the regularization setup so that your target depth is covered well, in that the regularization blocks to this depth are sufficiently small to resolve resistivity structure at that depth. Finally, you want the regularization to go to a half space at the bottom, basically one giant block.

Arguments::**station_locations**

[np.ndarray(n_stations)] array of station locations along a profile line in meters.

Key Words/Attributes	Description
air_key	letter associated with the value of air <i>default</i> is 0
air_value	value given to an air cell, <i>default</i> is 1E13
binding_offset	offset from the right side of the furthest left hand model block in meters. The regularization
cell_width	width of cells with in station area in meters <i>default</i> is 100
description	description of the model for the model file. <i>default</i> is 'simple inversion'
elevation_profile	elevation profile along the profile line. given as np.ndarray(nx, 2), where the elements are
mesh_fn	full path to mesh file.
mesh_values	letter values of each triangular mesh element if the cell is free value is ?
model_columns	
model_name	
model_rows	
min_block_width	[float] minimum model block width in meters, <i>default</i> is 2*cell_width
n_layers	number of vertical layers in mesh <i>default</i> is 90
num_free_param	[int] number of free parameters in the model. this is a tricky number to estimate apparent
num_layers	[int] number of regularization layers.

Table 5 – continued from

Key Words/Attributes	Description
num_x_pad_cells	number of horizontal padding cells outside the station area that will increase in size by
num_x_pad_small_cells	number of horizontal padding cells just outside the station area with width cell_width. This
num_z_pad_cells	number of vertical padding cells below z_target_depth down to z_bottom. default is 5
prejudice_fn	full path to prejudice file default is ‘none’
reg_basename	basename of regularization file (model file) default is ‘Occam2DModel’
reg_fn	full path to regularization file (model file) default is save_path/reg_basename
rel_station_locations	relative station locations within the mesh. The locations are relative to the center of the stations.
save_path	full path to save mesh and model file to. default is current working directory.
statics_fn	full path to static shift file Static shifts in occam may not work. default is ‘none’
station_locations	location of stations in meters, can be on a relative grid or in UTM.
trigger	[float] multiplier to merge model blocks at depth. A higher number increases the number of merged blocks.
x_grid	location of horizontal grid nodes in meters
x_nodes	relative spacing between grid nodes
x_pad_multiplier	horizontal padding cells will increase by this multiple out to the edge of the grid. default is 1.5
z1_layer	thickness of the first layer in the model. Should be at least 1/4 of the first skin depth default is 10 m
z_bottom	bottom depth of the model (m). Needs to be large enough to be 1D at the edge. default is 1000 m
z_grid	location of vertical nodes in meters
z_nodes	relative distance between vertical nodes in meters
z_target_depth	depth to deepest target of interest. Below this depth cells will be padded to z_bottom

Note

regularization does not work with topography yet. Having problems calculating the number of free parameters.

Example

```
>>> edi_path = r"/home/mt/edi_files"
>>> profile = occam2d.Profile(edi_path=edi_path)
>>> profile.generate_profile()
>>> reg = occam2d.Regularization(profile.station_locations)
>>> reg.build_mesh()
>>> reg.build_regularization()
>>> reg.save_path = r"/home/occam2d/Line1/Inv1"
>>> reg.write_regularization_file()
```

build_regularization()

Builds larger boxes around existing mesh blocks for the regularization.

As the model deepens the regularization boxes get larger.

The regularization boxes are merged mesh cells as prescribed by the Occam method.

get_num_free_params()

Estimate the number of free parameters in model mesh.

I'm assuming that if there are any fixed parameters in the block, then that model block is assumed to be fixed. Not sure if this is right cause there is no documentation.

DOES NOT WORK YET

read_regularization_file(*reg_fn*)

Read in a regularization file and populate attributes: * binding_offset * mesh_fn * model_columns * model_rows * prejudice_fn * statics_fn

write_regularization_file(*reg_fn=None*, *reg_basename=None*, *statics_fn='none'*, *prejudice_fn='none'*, *save_path=None*)

Write a regularization file for input into occam.

Calls build_regularization if build_regularization has not already been called.

if *reg_fn* is None, then file is written to *save_path/reg_basename*

Arguments::

reg_fn

[string] full path to regularization file. *default* is None and file will be written to *save_path/reg_basename*

reg_basename

[string] basename of regularization file

statics_fn

[string] full path to static shift file .. note:: static shift does not always work in
occam2d.exe

prejudice_fn

[string] full path to prejudice file

save_path

[string] path to save regularization file. *default* is current working directory

mtpy.modeling.occam2d.startup module

Created on Tue Mar 7 18:24:58 2023

@author: jpeacock

class mtpy.modeling.occam2d.startup.Startup(kwargs)**

Bases: object

Reads and writes the startup file for Occam2D.

 **Note**

Be sure to look at the Occam 2D documentation for description of all parameters

Key Words/Attributes	Description
data_fn	full path to data file
date_time	date and time the startup file was written
debug_level	[0 1 2] see occam documentation <i>default</i> is 1
description	brief description of inversion run <i>default</i> is ‘startup created by mtpy’
diagonal_penalties	penalties on diagonal terms <i>default</i> is 0
format	Occam file format <i>default</i> is ‘OCCAMITER_FLEX’
iteration	current iteration number <i>default</i> is 0
iterations_to_run	maximum number of iterations to run <i>default</i> is 20
lagrange_value	starting lagrange value <i>default</i> is 5
misfit_reached	[0 1] 0 if misfit has been reached, 1 if it has. <i>default</i> is 0
misfit_value	current misfit value. <i>default</i> is 1000
model_fn	full path to model file
model_limits	limits on model resistivity values <i>default</i> is None
model_value_step	limits on the step size of model values <i>default</i> is None
model_values	np.ndarray(num_free_params) of model values
param_count	number of free parameters in model
resistivity_start	starting resistivity value. If model_values is not given, then all values within model_values array will be set to resistivity_start
roughness_type	[0 1 2] type of roughness <i>default</i> is 1
roughness_value	current roughness value. <i>default</i> is 1E10
save_path	directory path to save startup file to <i>default</i> is current working directory
startup_basename	basename of startup file name. <i>default</i> is Occam2DStartup
startup_fn	full path to startup file. <i>default</i> is save_path/startup_basename
stepsize_count	max number of iterations per step <i>default</i> is 8
target_misfit	target misfit value. <i>default</i> is 1.

Example

```
>>> startup = occam2d.Startup()
>>> startup.data_fn = ocd.data_fn
>>> startup.model_fn = profile.reg_fn
>>> startup.param_count = profile.num_free_params
>>> startup.save_path = r"/home/occam2d/Line1/Inv1"
```

`write_startup_file(startup_fn=None, save_path=None, startup_basename=None)`

Write a startup file based on the parameters of startup class.

Default file name is save_path/startup_basename

Arguments::

startup_fn
 [string] full path to startup file. *default* is None

save_path
 [string] directory to save startup file. *default* is None

startup_basename
 [string] basename of startup file. *default* is None

Module contents

```
class mtpy.modeling.occam2d.Mesh(station_locations=None, **kwargs)
```

Bases: object

deals only with the finite element mesh. Builds a finite element mesh based on given parameters defined below. The mesh reads in the station locations, finds the center and makes the relative location of the furthest left hand station 0. The mesh increases in depth logarithmically as required by the physics of MT. Also, the model extends horizontally and vertically with padding cells in order to fulfill the assumption of the forward operator that at the edges the structure is 1D. Stations are placed on the horizontal nodes as required by Wannamaker's forward operator.

Mesh has the ability to create a mesh that incorporates topography given a elevation profile. It adds more cells to the mesh with thickness z1_layer. It then sets the values of the triangular elements according to the elevation value at that location. If the elevation covers less than 50% of the triangular cell, then the cell value is set to that of air

 **Note**

Mesh is inherited by Regularization, so the mesh can also be built from there, same as the example below.

Arguments:

Key Words/Attrib	Description
air_key	letter associated with the value of air <i>default</i> is 0
air_value	value given to an air cell, <i>default</i> is 1E13
cell_width	width of cells with in station area in meters <i>default</i> is 100
elevation_profile	elevation profile along the profile line. given as np.ndarray(nx, 2), where the elements are x_location, elevation. If elevation profile is given add_elevation is called automatically. <i>default</i> is None
mesh_fn	full path to mesh file.
mesh_values	letter values of each triangular mesh element if the cell is free value is ?
n_layers	number of vertical layers in mesh <i>default</i> is 90
num_x_pad_	number of horizontal padding cells outside the the station area that will increase in size by x_pad_multiplier. <i>default</i> is 7
num_x_pad_	number of horizontal padding cells just outside the station area with width cell_width. This is to extend the station area if needed. <i>default</i> is 2
num_z_pad_	number of vertical padding cells below z_target_depth down to z_bottom. <i>default</i> is 5
rel_station_lc	relative station locations within the mesh. The locations are relative to the center of the station area. <i>default</i> is None, filled later
save_path	full path to save mesh file to. <i>default</i> is current working directory.
sta-tion_location	location of stations in meters, can be on a relative grid or in UTM.
x_grid	location of horizontal grid nodes in meters
x_nodes	relative spacing between grid nodes
x_pad_multipl	horizontal padding cells will increase by this multiple out to the edge of the grid. <i>default</i> is 1.5
z1_layer	thickness of the first layer in the model. Should be at least 1/4 of the first skin depth <i>default</i> is 10
z_bottom	bottom depth of the model (m). Needs to be large enough to be 1D at the edge. <i>default</i> is 200000.0
z_grid	location of vertical nodes in meters
z_nodes	relative distance between vertical nodes in meters
z_target_dept	depth to deepest target of interest. Below this depth cells will be padded to z_bottom

Methods	Description
add_elevation	adds elevation to the mesh given elevation profile.
build_mesh	builds the mesh given the attributes of Mesh. If elevation_profile is not None, add_elevation is called inside build_mesh
plot_mesh	plots the built mesh with station location.
read_mesh_fil	reads in an existing mesh file and populates the appropriate attributes.
write_mesh_fi	writes a mesh file to save_path

Example

```
>>> import mtpy.modeling.occam2d as occcam2d
>>> edipath = r"/home/mt/edi_files"
>>> slist = ['mt{:0:03}'.format(ss) for ss in range(20)]
>>> ocd = occcam2d.Data(edi_path=edipath, station_list=slist)
>>> ocd.save_path = r"/home/occam/Line1/Inv1"
>>> ocd.write_data_file()
```

(continues on next page)

(continued from previous page)

```
>>> ocm = occam2d.Mesh(ocd.station_locations)
>>> # add in elevation
>>> ocm.elevation_profile = ocd.elevation_profile
>>> # change number of layers
>>> ocm.n_layers = 110
>>> # change cell width in station area
>>> ocm.cell_width = 200
>>> ocm.build_mesh()
>>> ocm.plot_mesh()
>>> ocm.save_path = ocd.save_path
>>> ocm.write_mesh_file()
```

add_elevation(elevation_profile=None)

the elevation model needs to be in relative coordinates and be a numpy.ndarray(2, num_elevation_points) where the first column is the horizontal location and the second column is the elevation at that location.

If you have a elevation model use Profile to project the elevation information onto the profile line

To build the elevation I'm going to add the elevation to the top of the model which will add cells to the mesh. there might be a better way to do this, but this is the first attempt. So I'm going to assume that the first layer of the mesh without elevation is the minimum elevation and blocks will be added to max elevation at an increment according to z1_layer

Note

the elevation model should be symmetrical ie, starting at the first station and ending on the last station, so for now any elevation outside the station area will be ignored and set to the elevation of the station at the extremities. This is not ideal but works for now.

Arguments:**elevation_profile**

[np.ndarray(2, num_elev_points)]

- 1st row is for profile location
- 2nd row is for elevation values

Computes:**mesh_values**

[mesh values, setting anything above topography] to the key for air, which for Occam is '0'

build_mesh()

Build the finite element mesh given the parameters defined by the attributes of Mesh. Computes relative station locations by finding the center of the station area and setting the middle to 0. Mesh blocks are built by calculating the distance between stations and putting evenly spaced blocks between the stations being close to cell_width. This places a horizontal node at the station location. If the spacing between stations is smaller than cell_width, a horizontal node is placed between the stations to be sure the model has room to change between the station.

If elevation_profile is given, add_elevation is called to add topography into the mesh.

Populates attributes:

- mesh_values
- rel_station_locations
- x_grid
- x_nodes
- z_grid
- z_nodes

Example

```
::      >>> import mtpy.modeling.occam2d as occcam2d >>> edipath =  
r"/home/mt/edi_files" >>> slist = ['mt{0:03}'.format(ss) for ss in range(20)]  
>>> ocd = occcam2d.Data(edipath=edipath, station_list=slist) >>> ocd.save_path  
= r"/home/occam/Line1/Inv1" >>> ocd.write_data_file() >>> ocm = occ-  
cam2d.Mesh(ocd.station_locations) >>> # add in elevation >>> ocm.elevation_profile  
= ocd.elevation_profile >>> # change number of layers >>> ocm.n_layers = 110 >>> #  
change cell width in station area >>> ocm.cell_width = 200 >>> ocm.build_mesh()
```

plot_mesh(kwargs)**

Plot built mesh with station locations.

Key Words	Description
depth_scale	[‘km’ ‘m’] scale of mesh plot. <i>default</i> is ‘km’
fig_dpi	dots-per-inch resolution of the figure <i>default</i> is 300
fig_num	number of the figure instance <i>default</i> is ‘Mesh’
fig_size	size of figure in inches (width, height) <i>default</i> is [5, 5]
fs	size of font of axis tick labels, axis labels are fs+2. <i>default</i> is 6
ls	[‘-’ ‘.’ ‘:’] line style of mesh lines <i>default</i> is ‘-’
marker	marker of stations <i>default</i> is r"\$lacktriangledown\$"
ms	size of marker in points. <i>default</i> is 5
plot_triangles	[‘y’ ‘n’] to plot mesh triangles. <i>default</i> is ‘n’

read_mesh_file(mesh_fn)

reads an occam2d 2D mesh file

Arguments:**mesh_fn**

[string] full path to mesh file

Populates:

x_grid : array of horizontal locations of nodes (m)

x_nodes: array of horizontal node relative distances
(column locations (m))

z_grid : array of vertical node locations (m)

z_nodes
[array of vertical nodes] (row locations(m))

mesh_values : np.array of free parameters

To do:

incorporate fixed values

Example

```
>>> import mtpy.modeling.occam2d as occam2d
>>> mg = occam2d.Mesh()
>>> mg.mesh_fn = r"/home/mt/occam/line1/Occam2Dmesh"
>>> mg.read_mesh_file()
```

write_mesh_file(*save_path=None*, *basename='Occam2DMesh'*)

Write a finite element mesh file.

Calls build_mesh if it already has not been called.

Arguments:

save_path

[string] directory path or full path to save file

basename

[string] basename of mesh file. *default* is ‘Occam2DMesh’

Returns:

mesh_fn

[string] full path to mesh file

example

```
>>> import mtpy.modeling.occam2d as occam2d
>>> edi_path = r"/home/mt/edi_files"
>>> profile = occam2d.Profile(edi_path)
>>> profile.plot_profile()
>>> mesh = occam2d.Mesh(profile.station_locations)
>>> mesh.build_mesh()
>>> mesh.write_mesh_file(save_path=r"/home/occam2d/Inv1")
```

class `mtpy.modeling.occam2d.Occam2DData`(*dataframe=None*, *center_point=None*, ***kwargs*)

Bases: object

Reads and writes data files and more.

Inherits Profile, so the intended use is to use Data to project stations onto a profile, then write the data file.

Model Modes	Description
1 or log_all	Log resistivity of TE and TM plus Tipper
2 or log_te_tip	Log resistivity of TE plus Tipper
3 or log_tm_tip	Log resistivity of TM plus Tipper
4 or log_te_tm	Log resistivity of TE and TM
5 or log_te	Log resistivity of TE
6 or log_tm	Log resistivity of TM
7 or all	TE, TM and Tipper
8 or te_tip	TE plus Tipper
9 or tm_tip	TM plus Tipper
10 or te_tm	TE and TM mode
11 or te	TE mode
12 or tm	TM mode
13 or tip	Only Tipper

Example Write Data File

```
>>> from mtpy.modeling.occam2d import Data
>>> occam_data_object = Data()
>>> occam_data_object.read_data_file(r"path/to/data/file.dat")
>>> occam_data_object.model_mode = 2
>>> occam_data_object.write_data_file(r"path/to/new/data/file_te.dat")
```

property data_filename

Data filename.

property dataframe

Dataframe function.

property frequencies

Frequencies function.

mask_from_datafile(mask_datafn)

Reads a separate data file and applies mask from this data file.

mask_datafn needs to have exactly the same frequencies, and station names must match exactly.

property n_data

N data.

property n_frequencies

N frequencies.

property n_stations

N stations.

property offsets

Offsets function.

read_data_file(data_fn=None)

Read in an existing data file and populate appropriate attributes

- data
- data_list

- freq
- station_list
- station_locations

Arguments::**data_fn**

[string] full path to data file *default* is None and set to save_path/fn_basename

Example

```
>>> import mtpy.modeling.occam2d as occam2d
>>> ocd = occam2d.Data()
>>> ocd.read_data_file(r"/home/Occam2D/Line1/Inv1/Data.dat")
```

property stations

Stations function.

write_data_file(data_fn=None)

Write a data file.

Arguments::**data_fn**

[string] full path to data file. *default* is save_path/fn_basename

If there data is None, then _fill_data is called to create a profile, rotate data and get all the necessary data. This way you can use write_data_file directly without going through the steps of projecting the stations, etc.

Example

```
:: >>> edipath = r"/home/mt/edi_files" >>> slst = ['mt{:0:03}'.format(ss) for ss in range(1, 20)] >>> ocd = occam2d.Data(edipath=edipath, station_list=slst) >>> ocd.save_path = r"/home/occam/line1/inv1" >>> ocd.write_data_file()
```

class mtpy.modeling.occam2d.Occam2DModel(iter_fn=None, model_fn=None, mesh_fn=None, **kwargs)

Bases: *Startup*

Read .iter file output by Occam2d. Builds the resistivity model from mesh and regularization files found from the .iter file. The resistivity model is an array(x_nodes, z_nodes) set on a regular grid, and the values of the model response are filled in according to the regularization grid. This allows for faster plotting.

Inherits Startup because they are basically the same object.

Argument:**iter_fn**

[string] full path to .iter file to read. *default* is None.

model_fn

[string] full path to regularization file. *default* is None and found directly from the .iter file.
Only input if the regularization is different from the file that is in the .iter file.

mesh_fn

[string] full path to mesh file. *default* is None Found directly from the model_fn file. Only input if the mesh is different from the file that is in the model file.

Key Words/Attributes	Description
data_fn	full path to data file
iter_fn	full path to .iter file
mesh_fn	full path to mesh file
mesh_x	np.ndarray(x_nodes, z_nodes) mesh grid for plotting
mesh_z	np.ndarray(x_nodes, z_nodes) mesh grid for plotting
model_values	model values from startup file
plot_x	nodes of mesh in horizontal direction
plot_z	nodes of mesh in vertical direction
res_model	np.ndarray(x_nodes, z_nodes) resistivity model values in linear scale

Methods	Description
build_model	get the resistivity model from the .iter file in a regular grid according to the mesh file with resistivity values according to the model file
read_iter_file	read .iter file and fill appropriate attributes
write_iter_file	write an .iter file incase you want to set it as the starting model or a priori model

Example

```
::      >>> model = occam2D.Model(r"/home/occam/line1/inv1/test_01.iter")    >>>
model.build_model()
```

build_model()

build the model from the mesh, regularization grid and model file

read_iter_file(iter_fn=None)

Read an iteration file.

Arguments:**iter_fn**

[string] full path to iteration file if iterpath=None. If iterpath is input then iterfn is just the name of the file without the full path.

Returns:**Example**

```
>>> import mtpy.modeling.occam2d as occam2d
>>> itfn = r"/home/0ccam2D/Line1/Inv1/Test_15.iter"
>>> ocm = occam2d.Model(itfn)
>>> ocm.read_iter_file()
```

write_iter_file(iter_fn=None)

write an iteration file if you need to for some reason, same as startup file

class mtpy.modeling.occam2d.Regularization(station_locations=None, **kwargs)

Bases: *Mesh*

Creates a regularization grid based on Mesh.

Note that Mesh is inherited

**by Regularization, therefore the intended use is to build a mesh with
the Regularization class.**

The regularization grid is what Occam calculates the inverse model on. Setup is tricky and can be painful, as you can see it is not quite fully functional yet, as it cannot incorporate topography yet. It seems like you'd like to have the regularization setup so that your target depth is covered well, in that the regularization blocks to this depth are sufficiently small to resolve resistivity structure at that depth. Finally, you want the regularization to go to a half space at the bottom, basically one giant block.

Arguments::

station_locations

[np.ndarray(n_stations)] array of station locations along a profile line in meters.

Key Words/Attributes	Description
air_key	letter associated with the value of air <i>default</i> is 0
air_value	value given to an air cell, <i>default</i> is 1E13
binding_offset	offset from the right side of the furthest left hand model block in meters. The regularization blocks to this depth are sufficiently small to resolve resistivity structure at that depth. Finally, you want the regularization to go to a half space at the bottom, basically one giant block.
cell_width	width of cells with in station area in meters <i>default</i> is 100
description	description of the model for the model file. <i>default</i> is ‘simple inversion’
elevation_profile	elevation profile along the profile line. given as np.ndarray(nx, 2), where the elements are full path to mesh file.
mesh_fn	letter values of each triangular mesh element if the cell is free value is ?
mesh_values	model_columns
model_name	model_rows
model_rows	min_block_width [float] minimum model block width in meters, <i>default</i> is 2*cell_width
n_layers	number of vertical layers in mesh <i>default</i> is 90
num_free_param	[int] number of free parameters in the model. this is a tricky number to estimate apparently
num_layers	[int] number of regularization layers.
num_x_pad_cells	number of horizontal padding cells outside the the station area that will increase in size by
num_x_pad_small_cells	number of horizontal padding cells just outside the station area with width cell_width. This is a trick to make sure the model is centered in the grid.
num_z_pad_cells	number of vertical padding cells below z_target_depth down to z_bottom. <i>default</i> is 5
prejudice_fn	full path to prejudice file <i>default</i> is ‘none’
reg_basename	basename of regularization file (model file) <i>default</i> is ‘Occam2DModel’
reg_fn	full path to regularization file (model file) <i>default</i> is save_path/reg_basename
rel_station_locations	relative station locations within the mesh. The locations are relative to the center of the station area.
save_path	full path to save mesh and model file to. <i>default</i> is current working directory.
statics_fn	full path to static shift file Static shifts in occam may not work. <i>default</i> is ‘none’
station_locations	location of stations in meters, can be on a relative grid or in UTM.
trigger	[float] multiplier to merge model blocks at depth. A higher number increases the number of blocks.
x_grid	location of horizontal grid nodes in meters
x_nodes	relative spacing between grid nodes
x_pad_multiplier	horizontal padding cells will increase by this multiple out to the edge of the grid. <i>default</i> is 1.5
z1_layer	thickness of the first layer in the model. Should be at least 1/4 of the first skin depth <i>default</i> is 100 meters
z_bottom	bottom depth of the model (m). Needs to be large enough to be 1D at the edge. <i>default</i> is 1000 meters
z_grid	location of vertical nodes in meters
z_nodes	relative distance between vertical nodes in meters
z_target_depth	depth to deepest target of interest. Below this depth cells will be padded to z_bottom

Note

regularization does not work with topography yet. Having problems calculating the number of

free parameters.

Example

```
>>> edi_path = r"/home/mt/edi_files"
>>> profile = occam2d.Profile(edi_path=edi_path)
>>> profile.generate_profile()
>>> reg = occam2d.Regularization(profile.station_locations)
>>> reg.build_mesh()
>>> reg.build_regularization()
>>> reg.save_path = r"/home/occam2d/Line1/Inv1"
>>> reg.write_regularization_file()
```

build_regularization()

Builds larger boxes around existing mesh blocks for the regularization.

As the model deepens the regularization boxes get larger.

The regularization boxes are merged mesh cells as prescribed by the Occam method.

get_num_free_params()

Estimate the number of free parameters in model mesh.

I'm assuming that if there are any fixed parameters in the block, then that model block is assumed to be fixed. Not sure if this is right cause there is no documentation.

DOES NOT WORK YET

read_regularization_file(*reg_fn*)

Read in a regularization file and populate attributes: * binding_offset * mesh_fn * model_columns * model_rows * prejudice_fn * statics_fn

write_regularization_file(*reg_fn=None*, *reg_basename=None*, *statics_fn='none'*, *prejudice_fn='none'*, *save_path=None*)

Write a regularization file for input into occam.

Calls build_regularization if build_regularization has not already been called.

if *reg_fn* is None, then file is written to *save_path/reg_basename*

Arguments::

reg_fn

[string] full path to regularization file. *default* is None and file will be written to *save_path/reg_basename*

reg_basename

[string] basename of regularization file

statics_fn

[string] full path to static shift file .. note:: static shift does not always work in
occam2d.exe

prejudice_fn

[string] full path to prejudice file

save_path

[string] path to save regularization file. *default* is current working directory

```
class mtpy.modeling.occam2d.Startup(**kwargs)
```

Bases: object

Reads and writes the startup file for Occam2D.

Note

Be sure to look at the Occam 2D documentation for description of all parameters

Key Words/Attributes	Description
data_fn	full path to data file
date_time	date and time the startup file was written
debug_level	[0 1 2] see occam documentation <i>default</i> is 1
description	brief description of inversion run <i>default</i> is ‘startup created by mtpy’
diagonal_penalties	penalties on diagonal terms <i>default</i> is 0
format	Occam file format <i>default</i> is ‘OCCAMITER_FLEX’
iteration	current iteration number <i>default</i> is 0
iterations_to_run	maximum number of iterations to run <i>default</i> is 20
lagrange_value	starting lagrange value <i>default</i> is 5
misfit_reached	[0 1] 0 if misfit has been reached, 1 if it has. <i>default</i> is 0
misfit_value	current misfit value. <i>default</i> is 1000
model_fn	full path to model file
model_limits	limits on model resistivity values <i>default</i> is None
model_value_step	limits on the step size of model values <i>default</i> is None
model_values	np.ndarray(num_free_params) of model values
param_count	number of free parameters in model
resistivity_start	starting resistivity value. If model_values is not given, then all values with in model_values array will be set to resistivity_start
roughness_type	[0 1 2] type of roughness <i>default</i> is 1
roughness_value	current roughness value. <i>default</i> is 1E10
save_path	directory path to save startup file to <i>default</i> is current working directory
startup_basename	basename of startup file name. <i>default</i> is Occam2DStartup
startup_fn	full path to startup file. <i>default</i> is save_path/startup_basename
stepsize_count	max number of iterations per step <i>default</i> is 8
target_misfit	target misfit value. <i>default</i> is 1.

Example

```
>>> startup = occam2d.Startup()
>>> startup.data_fn = ocd.data_fn
>>> startup.model_fn = profile.reg_fn
>>> startup.param_count = profile.num_free_params
>>> startup.save_path = r"/home/occam2d/Line1/Inv1"
```

`write_startup_file(startup_fn=None, save_path=None, startup_basename=None)`

Write a startup file based on the parameters of startup class.

Default file name is save_path/startup_basename

Arguments::

startup_fn

[string] full path to startup file. *default* is None

save_path

[string] directory to save startup file. *default* is None

startup_basename

[string] basename of startup file. *default* is None

mtpy.modeling.plots package

Submodules

mtpy.modeling.plots.plot_mesh module

Created on Fri Oct 14 08:37:48 2022

@author: jpeacock

class mtpy.modeling.plots.plot_mesh.PlotMesh(*model_obj*, ***kwargs*)

Bases: *PlotBase*

plot()

Plot the mesh to show model grid.

Arguments:

```
**z_limits** : tuple (zmin,zmax)
    plot min and max distances in meters for the
    vertical direction. If None, the z_limits is
    set to the number of layers. Z is positive down
    *default* is None
```

mtpy.modeling.plots.plot_modem_rms module

Created on Wed Feb 17 10:57:29 2021

copyright

Jared Peacock (jpeacock@usgs.gov)

license

MIT

class mtpy.modeling.plots.plot_modem_rms.PlotRMS(*dataframe*, ***kwargs*)

Bases: *PlotBaseMaps*

property dataframe

Dataframe function.

plot(kwargs)**

Plot function. :param ***kwargs*: DESCRIPTION. :type ***kwargs*: TYPE :return: DESCRIPTION. :rtype: TYPE

print_suspect_stations(*rms_threshold=4*)

Print stations that are suspect. :return: DESCRIPTION. :rtype: TYPE

```
property rms_array
    Arrays for color maps. :return: DESCRIPTION. :rtype: TYPE

property rms_cmap
    Rms cmap.

property rms_per_period_all
    RMS per period.

property rms_per_station
    RMS per period.
```

Module contents

```
class mtpy.modeling.plots.PlotMesh(model_obj, **kwargs)
```

Bases: *PlotBase*

```
plot()
```

Plot the mesh to show model grid.

Arguments:

```
**z_limits** : tuple (zmin,zmax)
    plot min and max distances in meters for the
    vertical direction. If None, the z_limits is
    set to the number of layers. Z is positive down
    *default* is None
```

```
class mtpy.modeling.plots.PlotRMS(dataframe, **kwargs)
```

Bases: *PlotBaseMaps*

```
property dataframe
```

Dataframe function.

```
plot(**kwargs)
```

Plot function. :param **kwargs: DESCRIPTION. :type **kwargs: TYPE :return: DESCRIPTION. :rtype: TYPE

```
print_suspect_stations(rms_threshold=4)
```

Print stations that are suspect. :return: DESCRIPTION. :rtype: TYPE

```
property rms_array
```

Arrays for color maps. :return: DESCRIPTION. :rtype: TYPE

```
property rms_cmap
```

Rms cmap.

```
property rms_per_period_all
```

RMS per period.

```
property rms_per_station
```

RMS per period.

mtpy.modeling.simpeg package**Subpackages****mtpy.modeling.simpeg.recipes package****Submodules****mtpy.modeling.simpeg.recipes.inversion_1d module**

Created on Wed Nov 1 11:58:59 2023

@author: jpeacock

class mtpy.modeling.simpeg.recipes.inversion_1d.**Simpeg1D**(*mt_dataframe=None*, ***kwargs*)

Bases: object

Run a 1D simpeg inversion.

cull_from_difference(*sub_df*, *max_diff_res=1.0*, *max_diff_phase=10*)

Remove points based on a simple difference between neighboring points

uses np.diff

res difference is in log space. :param sub_df: :param max_diff_res: DESCRIPTION, defaults to 1.0. :type max_diff_res: TYPE, optional :param max_diff_phase: DESCRIPTION, defaults to 10. :type max_diff_phase: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

cull_from_interpolated(*sub_df*, *tolerance=0.1*, *s_factor=2*)

create a cubic spline as a smooth version of the data and then find points a certain distance away to remove.

:param : DESCRIPTION :type : TYPE :return: DESCRIPTION :rtype: TYPE

cull_from_model(*iteration*)

Remove bad point based on initial run. :param iteration: DESCRIPTION. :type iteration: TYPE :return: DESCRIPTION. :rtype: TYPE

property data

Data function.

property data_error

Data error.

property frequencies

Frequencies function.

property mesh

Mesh function.

property mode

Mode function.

property periods

Periods function.

plot_model_fitting(*scale='log'*, *fig_num=1*)

Plot predicted vs model. :return: DESCRIPTION. :rtype: TYPE

plot_response(*iteration=None, fig_num=2*)

Plot response. :param fig_num:

Defaults to 2.

Parameters

iteration (*TYPE, optional*) – DESCRIPTION, defaults to None.

Returns

DESCRIPTION.

Return type

TYPE

run_fixed_layer_inversion(*cull_from_difference=True, maxIter=40, maxIterCG=30, alpha_s=1e-10, alpha_z=1, beta0_ratio=1, coolingFactor=2, coolingRate=1, chi_factor=1, use_irls=False, p_s=2, p_z=2*)

Run fixed layer inversion.

property thicknesses

Thicknesses function.

Module contents

Created on Wed Nov 1 11:58:39 2023

@author: jpeacock

class `mtpy.modeling.simpeg.recipes.Simpeg1D`(*mt_dataframe=None, **kwargs*)

Bases: `object`

Run a 1D simpeg inversion.

cull_from_difference(*sub_df, max_diff_res=1.0, max_diff_phase=10*)

Remove points based on a simple difference between neighboring points

uses `np.diff`

res difference is in log space. :param sub_df: :param max_diff_res: DESCRIPTION, defaults to 1.0. :type max_diff_res: *TYPE*, optional :param max_diff_phase: DESCRIPTION, defaults to 10. :type max_diff_phase: *TYPE*, optional :return: DESCRIPTION. :rtype: *TYPE*

cull_from_interpolated(*sub_df, tolerance=0.1, s_factor=2*)

create a cubic spline as a smooth version of the data and then find points a certain distance away to remove.

:param : DESCRIPTION :type : *TYPE* :return: DESCRIPTION :rtype: *TYPE*

cull_from_model(*iteration*)

Remove bad point based on initial run. :param iteration: DESCRIPTION. :type iteration: *TYPE* :return: DESCRIPTION. :rtype: *TYPE*

property data

Data function.

property data_error

Data error.

property frequencies

Frequencies function.

property mesh

Mesh function.

property mode

Mode function.

property periods

Periods function.

plot_model_fitting(*scale='log'*, *fig_num=1*)

Plot predicted vs model. :return: DESCRIPTION. :rtype: TYPE

plot_response(*iteration=None*, *fig_num=2*)

Plot response. :param fig_num:

Defaults to 2.

Parameters

iteration (TYPE, optional) – DESCRIPTION, defaults to None.

Returns

DESCRIPTION.

Return type

TYPE

**run_fixed_layer_inversion(*cull_from_difference=True*, *maxIter=40*, *maxIterCG=30*, *alpha_s=1e-10*,
alpha_z=1, *beta0_ratio=1*, *coolingFactor=2*, *coolingRate=1*, *chi_factor=1*,
use_irls=False, *p_s=2*, *p_z=2*)**

Run fixed layer inversion.

property thicknesses

Thicknesses function.

**class mtpy.modeling.simpeg.recipes.Simpeg2D(*dataframe*, *data_kwargs={}*, *mesh_kwargs={}*,
mesh_type='tensor', ***kwargs*)**

Bases: object

A vanilla recipe to invert 2D MT data.

- For now the default is a quad tree mesh
- Optimization: Inexact Gauss Newton
- Regularization: Sparse

change mesh to tensor mesh.

property active_map

Active cells mapping

Returns

DESCRIPTION

Return type

TYPE

property beta_schedule

how quickly beta is reduced

Returns
DESCRIPTION

Return type
TYPE

property conductivity_map
conductivity mapping

Returns
DESCRIPTION

Return type
TYPE

property data_misfit
data misfit of all components TE + TM

property directives
list of directives to supply to the inversion

Returns
DESCRIPTION

Return type
TYPE

property exponent_map
compute fields on an exponential mapping :return: DESCRIPTION :rtype: TYPE

property inverse_problem
setup the inverse problem

Returns
DESCRIPTION

Return type
TYPE

property iterations
return dictionary of model outputs

make_mesh(kwargs)**
make QuadTree Mesh

property optimization
optimization algorithm
default is InexactGaussNewton

plot_iteration(iteration_number, resistivity=True, **kwargs)

plot_responses(iteration_number, **kwargs)
Plot responses all together

Parameters

- **iteration** (TYPE) – DESCRIPTION
- ****kwargs** – DESCRIPTION

Returns
DESCRIPTION

Return type
TYPE

plot_tikhonov_curve()
plot L-like curve

Returns
DESCRIPTION

Return type
TYPE

property reference_model
reference model

Returns
DESCRIPTION

Return type
TYPE

property regularization
Create sparse regularization using paramaters

- alpha_s = smallness parameter
- alpha_y = smoothing in y direction
- alpha_z = smoothing in z direction

Returns
DESCRIPTION

Return type
TYPE

run_inversion()
run the inversion using the attributes as input.

property starting_beta
set up the starting beta value

Returns
DESCRIPTION

Return type
TYPE

property target_misfit
target misfit

Returns
DESCRIPTION

Return type
TYPE

property te_data_misfit
data misfit of TE mode

```
property te_simulation
    Simulation for TE Mode

property tm_data_misfit
    data misfit of TM mode

property tm_simulation
    Simulation for TE Mode
```

Submodules

mtpy.modeling.simpeg.make_2d_mesh module

Created on Thu Nov 9 10:43:06 2023

@author: jpeacock

```
class mtpy.modeling.simpeg.make_2d_mesh.QuadTreeMesh(station_locations, frequencies, **kwargs)
```

Bases: object

build a quad tree mesh based on station locations and frequencies to invert

dimensions are x for the lateral dimension and z for the vertical.

station locations should be offsets from a single point [offset, elevation]. should be shape [n, 2]
topography should be [x, z] :- [n, 2] and in station location coordinate system.

```
property active_cell_index
```

return active cell mask

TODO: include topographic surface

Returns

DESCRIPTION

Return type

TYPE

```
property dx
```

```
property dz
```

```
make_mesh(**kwargs)
```

create mesh

```
property number_of_active_cells
```

number of active cells

```
property nx
```

```
property nz
```

```
plot_mesh(**kwargs)
```

plot the mesh

```
property x_pad
```

```
property x_total
```

get the total distance in the horizontal direction.

```
property z_core
property z_max
property z_pad_down
property z_pad_up
property z_total

class mtpy.modeling.simpeg.make_2d_mesh.StructuredMesh(station_locations, frequencies, **kwargs)
    Bases: object

    property active_cell_index
        return active cell mask
        TODO: include topographic surface

        Returns
        DESCRIPTION

        Return type
        TYPE

    property dx
    property frequency_max
    property frequency_min
    make_mesh()
        create structured mesh

        Returns
        DESCRIPTION

        Return type
        TYPE

    property n_station_x_cells
    property n_x_padding
    property number_of_active_cells
        number of active cells
    plot_mesh(**kwargs)
        plot the mesh
    property station_total_length
    property x_padding_cells
    property z1_layer_thickness
    property z_bottom
    property z_mesh_down
    property z_mesh_up
```

Module contents

Created on Wed Nov 1 11:58:31 2023

@author: jpeacock

mtpy.modeling.ws3dinv package

Submodules

mtpy.modeling.ws3dinv.data module

Merge transfer functions together

`class mtpy.modeling.ws3dinv.data.WSData(mt_dataframe=None, **kwargs)`

Bases: object

Includes tools for reading and writing data files intended to be used with ws3dinv.

Example

```
>>> import mtpy.modeling.ws3dinv as ws
>>> import os
>>> edi_path = r"/home/EDI_Files"
>>> edi_list = [os.path.join(edi_path, edi) for edi in edi_path
>>> ...           if edi.find('.edi') > 0]
>>> # create an evenly space period list in log space
>>> p_list = np.logspace(np.log10(.001), np.log10(1000), 12)
>>> wsdata = ws.WSData(edi_list=edi_list, period_list=p_list,
>>> ...                           station_fn=r"/home/stations.txt")
>>> wsdata.write_data_file()
```

Attributes	Description
data	<p>numpy structured array with keys:</p> <ul style="list-style-type: none"> • <i>station</i> → station name • <i>east</i> → relative eastern location in grid • <i>north</i> → relative northern location in grid • <i>z_data</i> → impedance tensor array with shape (n_stations, n_freq, 4, dtype=complex) • *z_data_err → impedance tensor error without error map applied • *z_err_map → error map from data file
data_fn	full path to data file
edi_list	list of edi files used to make data file
n_z	[4 8] number of impedance tensor elements <i>default</i> is 8
ncol	number of columns in out file from winglink <i>default</i> is 5
period_list	list of periods to invert for
ptol	if periods in edi files don't match period_list then program looks for periods within ptol <i>default</i> is .15 or 15 percent
rotation_angle	Angle to rotate the data relative to north. Here the angle is measure clockwise from North, Assuming North is 0 and East is 90. Rotating data, and grid to align with regional geoelectric strike can improve the inversion. <i>default</i> is None
save_path	path to save the data file
station_fn	full path to station file written by WSStation
station_locations	<p>numpy structured array for station locations keys:</p> <ul style="list-style-type: none"> • <i>station</i> → station name • <i>east</i> → relative eastern location in grid • <i>north</i> → relative northern location in grid
station_east	if input a station file is written
station_north	relative locations of station in east direction
station_names	relative locations of station in north direction
units	names of stations
wl_out_fn	['mv' 'else'] units of Z, needs to be mv for ws3dinv. <i>default</i> is 'mv'
wl_site_fn	Winglink .out file which describes a 3D grid
z_data	Wingling .sites file which gives station locations
z_data_err	impedance tensors of data with shape: (n_station, n_periods, 2, 2)
	error of data impedance tensors with error map applied, <u>shape (n_stations, n_periods, 2, 2)</u>
1.8. mtpyr	[float 'data'] 'data' to set errors as data errors or g 293 a percent error to impedance tensor elements <i>default</i> is .05 or 5% if given as percent, ie. 5% then it is converted to .05.

Methods	Description
build_data	builds the data from .edi files
write_data_file	writes a data file from attribute data. This way you can read in a data file, change some parameters and rewrite.
read_data_file	reads in a ws3dinv data file

property data_filename

Data filename.

property dataframe

Dataframe function.

get_n_stations()

Get n stations.

get_period_df(period)

Get period df.

property period

Period function.

read_data_file(data_filename)

Read in data file.

Arguments::**data_fn**

[string] full path to data file

wl_sites_fn

[string] full path to sites file output by winglink. This is to match the station name with station number.

station_fn

[string] full path to station location file written by WSStation

Fills Attributes::**data**

[structure np.ndarray] fills the attribute WSData.data with values

period_list

[np.ndarray()] fills the period list with values.

write_data_file(kwargs)**

Writes a data file based on the attribute data.

Key Word Arguments::**data_fn**

[string] full path to data file name

save_path

[string] directory path to save data file, will be written as save_path/data_basename

data_basename

[string] basename of data file to be saved as save_path/data_basename *default* is WS-DataFile.dat

Note

if any of the data attributes have been reset, be sure to call build_data() before write_data_file.

mtpy.modeling.ws3dinv.startup module

Created on Tue Nov 7 09:33:52 2023

@author: jpeacock

class mtpy.modeling.ws3dinv.startup.**WSStartup**(*data_fn=None, initial_fn=None, **kwargs*)

Bases: object

Read and write startup files

Example

```
>>> import mtpy.modeling.ws3dinv as ws
>>> dfn = r"/home/MT/ws3dinv/Inv1/WSDataFile.dat"
>>> ifn = r"/home/MT/ws3dinv/Inv1/init3d"
>>> sws = ws.WSStartup(data_fn=dfn, initial_fn=ifn)
```

apriori_fn full path to *a priori* model file

default is ‘default’

control_fn full path to model index control file

default is ‘default’

data_fn full path to data file **error_tol** error tolerance level

default is ‘default’

initial_fn full path to initial model file **lagrange** starting lagrange multiplier

default is ‘default’

max_iter max number of iterations

default is 10

model_ls model length scale

default is 5 0.3 0.3 0.3

output_stem output file name stem

default is ‘ws3dinv’

save_path directory to save file to **startup_fn** full path to startup file **static_fn** full path to statics file

default is ‘default’

target_rms target rms

default is 1.0

read_startup_file(*startup_fn*)

Read startup file fills attributes.

property startup_fn

Startup fn.

write_startup_file(save_path)

Makes a startup file for WSINV3D.

mtpy.modeling.ws3dinv.stations module

Created on Tue Nov 7 09:18:17 2023

@author: jpeacock

class mtpy.modeling.ws3dinv.stations.WSStation(station_fn=None, **kwargs)

Bases: object

Read and write a station file where the locations are relative to the 3D mesh.

Attributes	Description
east	array of relative locations in east direction
elev	array of elevations for each station
names	array of station names
north	array of relative locations in north direction
station_fn	full path to station file
save_path	path to save file to

Methods	Description
read_station_file	reads in a station file
write_station_file	writes a station file
write_vtk_file	writes a vtk points file for station locations

from_wl_write_station_file(sites_file, out_file, ncol=5)

Write a ws station file from the outputs of winglink.

Arguments::**sites_fn**

[string] full path to sites file output from winglink

out_fn

[string] full path to .out file output from winglink

ncol

[int] number of columns the data is in *default* is 5

read_station_file(station_filename)

Read in station file written by write_station_file.

Arguments::**station_fn**

[string] full path to station file

Outputs::**east**

[np.ndarray(n_stations)] relative station locations in east direction

north
[np.ndarray(n_stations)] relative station locations in north direction

elev
[np.ndarray(n_stations)] relative station locations in vertical direction

station_list
[list or np.ndarray(n_stations)] name of stations

property station_filename
Station filename.

write_station_file(east=None, north=None, station_list=None, save_path=None, elev=None)
Write a station file to go with the data file.
the locations are on a relative grid where (0, 0, 0) is the center of the grid. Also, the stations are assumed to be in the center of the cell.

Arguments::

east
[np.ndarray(n_stations)] relative station locations in east direction

north
[np.ndarray(n_stations)] relative station locations in north direction

elev
[np.ndarray(n_stations)] relative station locations in vertical direction

station_list
[list or np.ndarray(n_stations)] name of stations

save_path
[string] directory or full path to save station file to if a directory the file will be saved as save_path/WS_Station_Locations.txt if save_path is none the current working directory is used as save_path

Outputs::

station_fn : full path to station file

write_vtk_file(save_path, vtk_basename='VTKStations')

Write a vtk file to plot stations.

Arguments::

save_path
[string] directory to save file to. Will save as save_path/vtk_basename

vtk_basename
[string] base file name for vtk file, extension is automatically added.

Module contents

Created on Tue Nov 7 11:42:53 2023

@author: jpeacock

class mtpy.modeling.ws3dinv.WSData(mt_dataframe=None, **kwargs)

Bases: object

Includes tools for reading and writing data files intended to be used with ws3dinv.

Example

```
>>> import mtpy.modeling.ws3dinv as ws
>>> import os
>>> edi_path = r"/home/EDI_Files"
>>> edi_list = [os.path.join(edi_path, edi) for edi in edi_path
>>> ...           if edi.find('.edi') > 0]
>>> # create an evenly space period list in log space
>>> p_list = np.logspace(np.log10(.001), np.log10(1000), 12)
>>> wsdata = ws.WSData(edi_list=edi_list, period_list=p_list,
>>> ...                           station_fn=r"/home/stations.txt")
>>> wsdata.write_data_file()
```

Attributes	Description
data	<p>numpy structured array with keys:</p> <ul style="list-style-type: none"> • <i>station</i> → station name • <i>east</i> → relative eastern location in grid • <i>north</i> → relative northern location in grid • <i>z_data</i> → impedance tensor array with shape (n_stations, n_freq, 4, dtype=complex) • *z_data_err → impedance tensor error without error map applied • *z_err_map → error map from data file
data_fn	full path to data file
edi_list	list of edi files used to make data file
n_z	[4 8] number of impedance tensor elements <i>default</i> is 8
ncol	number of columns in out file from winglink <i>default</i> is 5
period_list	list of periods to invert for
ptol	if periods in edi files don't match period_list then program looks for periods within ptol <i>default</i> is .15 or 15 percent
rotation_angle	Angle to rotate the data relative to north. Here the angle is measure clockwise from North, Assuming North is 0 and East is 90. Rotating data, and grid to align with regional geoelectric strike can improve the inversion. <i>default</i> is None
save_path	path to save the data file
station_fn	full path to station file written by WSStation
station_locations	<p>numpy structured array for station locations keys:</p> <ul style="list-style-type: none"> • <i>station</i> → station name • <i>east</i> → relative eastern location in grid • <i>north</i> → relative northern location in grid
station_east	if input a station file is written
station_north	relative locations of station in east direction
station_names	relative locations of station in north direction
units	names of stations
wl_out_fn	['mv' 'else'] units of Z, needs to be mv for ws3dinv. <i>default</i> is 'mv'
wl_site_fn	Winglink .out file which describes a 3D grid
z_data	Wingling .sites file which gives station locations
z_data_err	impedance tensors of data with shape: (n_station, n_periods, 2, 2)
	error of data impedance tensors with error map applied, <u>shape (n_stations, n_periods, 2, 2)</u>
1.8. mtpyr	[float 'data'] 'data' to set errors as data errors or g 299 a percent error to impedance tensor elements <i>default</i> is .05 or 5% if given as percent, ie. 5% then it is converted to .05.

Methods	Description
build_data	builds the data from .edi files
write_data_file	writes a data file from attribute data. This way you can read in a data file, change some parameters and rewrite.
read_data_file	reads in a ws3dinv data file

property data_filename

Data filename.

property dataframe

Dataframe function.

get_n_stations()

Get n stations.

get_period_df(period)

Get period df.

property period

Period function.

read_data_file(data_filename)

Read in data file.

Arguments::**data_fn**

[string] full path to data file

wl_sites_fn

[string] full path to sites file output by winglink. This is to match the station name with station number.

station_fn

[string] full path to station location file written by WSStation

Fills Attributes::**data**

[structure np.ndarray] fills the attribute WSData.data with values

period_list

[np.ndarray()] fills the period list with values.

write_data_file(kwargs)**

Writes a data file based on the attribute data.

Key Word Arguments::**data_fn**

[string] full path to data file name

save_path

[string] directory path to save data file, will be written as save_path/data_basename

data_basename

[string] basename of data file to be saved as save_path/data_basename *default* is WS-DataFile.dat

Note

if any of the data attributes have been reset, be sure to call `build_data()` before `write_data_file`.

```
class mtpy.modeling.ws3dinv.WSStartup(data_fn=None, initial_fn=None, **kwargs)
```

Bases: `object`

Read and write startup files

Example

```
>>> import mtpy.modeling.ws3dinv as ws
>>> dfn = r"/home/MT/ws3dinv/Inv1/WSDataFile.dat"
>>> ifn = r"/home/MT/ws3dinv/Inv1/init3d"
>>> sws = ws.WSStartup(data_fn=dfn, initial_fn=ifn)
```

apriori_fn full path to *a priori* model file

default is ‘default’

control_fn full path to model index control file

default is ‘default’

data_fn full path to data file error_tol error tolerance level

default is ‘default’

initial_fn full path to initial model file lagrange starting lagrange multiplier

default is ‘default’

max_iter max number of iterations

default is 10

model_ls model length scale

default is 5 0.3 0.3 0.3

output_stem output file name stem

default is ‘ws3dinv’

save_path directory to save file to startup_fn full path to startup file static_fn full path to statics file

default is ‘default’

target_rms target rms

default is 1.0

read_startup_file(*startup_fn*)

Read startup file fills attributes.

property startup_fn

Startup fn.

write_startup_file(*save_path*)

Makes a startup file for WSINV3D.

```
class mtpy.modeling.ws3dinv.WSStation(station_fn=None, **kwargs)
```

Bases: object

Read and write a station file where the locations are relative to the 3D mesh.

Attributes	Description
east	array of relative locations in east direction
elev	array of elevations for each station
names	array of station names
north	array of relative locations in north direction
station_fn	full path to station file
save_path	path to save file to

Methods	Description
read_station_file	reads in a station file
write_station_file	writes a station file
write_vtk_file	writes a vtk points file for station locations

```
from_wl_write_station_file(sites_file, out_file, ncol=5)
```

Write a ws station file from the outputs of winglink.

Arguments::

sites_fn

[string] full path to sites file output from winglink

out_fn

[string] full path to .out file output from winglink

ncol

[int] number of columns the data is in *default* is 5

```
read_station_file(station_filename)
```

Read in station file written by write_station_file.

Arguments::

station_fn

[string] full path to station file

Outputs::

east

[np.ndarray(n_stations)] relative station locations in east direction

north

[np.ndarray(n_stations)] relative station locations in north direction

elev

[np.ndarray(n_stations)] relative station locations in vertical direction

station_list

[list or np.ndarray(n_stations)] name of stations

```
property station_filename
```

Station filename.

`write_station_file`(*east=None*, *north=None*, *station_list=None*, *save_path=None*, *elev=None*)

Write a station file to go with the data file.

the locations are on a relative grid where (0, 0, 0) is the center of the grid. Also, the stations are assumed to be in the center of the cell.

Arguments::**east**

[np.ndarray(n_stations)] relative station locations in east direction

north

[np.ndarray(n_stations)] relative station locations in north direction

elev

[np.ndarray(n_stations)] relative station locations in vertical direction

station_list

[list or np.ndarray(n_stations)] name of stations

save_path

[string] directory or full path to save station file to if a directory the file will be saved as save_path/WS_Station_Locations.txt if save_path is none the current working directory is used as save_path

Outputs::

station_fn : full path to station file

`write_vtk_file`(*save_path*, *vtk_basename='VTKStations'*)

Write a vtk file to plot stations.

Arguments::**save_path**

[string] directory to save file to. Will save as save_path/vtk_basename

vtk_basename

[string] base file name for vtk file, extension is automatically added.

Submodules**`mtpy.modeling.errors module`**

Created on Tue Oct 11 16:01:37 2022

@author: jpeacock

`class mtpy.modeling.errorsModelError`(*data=None*, *measurement_error=None*, *kwargs*)**

Bases: object

`compute_absolute_error()`

Compute absolute error. :param data: DESCRIPTION. :type data: TYPE :param error_value: DESCRIPTION. :type error_value: TYPE :return: DESCRIPTION. :rtype: TYPE

`compute_arithmetic_mean_error()`

Error_value * (Zxy + Zyx) / 2. :param data: DESCRIPTION. :type data: TYPE :param error_value: DESCRIPTION. :type error_value: TYPE :return: DESCRIPTION. :rtype: TYPE

`compute_eigen_value_error()`

Error_value * eigen(data).mean(). :param data: DESCRIPTION. :type data: TYPE :param error_value: DESCRIPTION. :type error_value: TYPE :return: DESCRIPTION. :rtype: TYPE

compute_error(*data=None*, *error_type=None*, *error_value=None*, *floor=None*)

Compute error. :param data: DESCRIPTION, defaults to None. :type data: TYPE, optional :param error_type: DESCRIPTION, defaults to None. :type error_type: TYPE, optional :param error_value: DESCRIPTION, defaults to None. :type error_value: TYPE, optional :param floor: DESCRIPTION, defaults to None. :type floor: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

compute_geometric_mean_error()

Error_value * sqrt(Zxy * Zyx). :param data: DESCRIPTION. :type data: TYPE :param error_value: DESCRIPTION. :type error_value: TYPE :return: DESCRIPTION. :rtype: TYPE

compute_median_error()

Median(array) * error_value. :param array: DESCRIPTION. :type array: TYPE :param error_value: DESCRIPTION. :type error_value: TYPE :return: DESCRIPTION. :rtype: TYPE

compute_percent_error()

Percent error. :param data: DESCRIPTION. :type data: TYPE :param percent: DESCRIPTION. :type percent: TYPE :return: DESCRIPTION. :rtype: TYPE

compute_row_error()

Set zxx and xzy the same error and zyy and zyx the same error. :param data: DESCRIPTION. :type data: TYPE :param error_value: DESCRIPTION. :type error_value: TYPE :param floor: DESCRIPTION, defaults to True. :type floor: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

property data

Data function.

property error_parameters

Error parameters.

property error_type

Error type.

property error_value

Error value.

property floor

Floor function.

mask_zeros(*data*)

Mask zeros. :param data: DESCRIPTION. :type data: TYPE :return: DESCRIPTION. :rtype: TYPE

property measurement_error

Measurement error.

property mode

Mode function.

resize_output(*error_array*)

Resize the error estimation to the same size as the input data. :param error_array: DESCRIPTION. :type error_array: TYPE :return: DESCRIPTION. :rtype: TYPE

set_floor(*error_array*)

Set error floor. :param error_array: :param array: DESCRIPTION. :type array: TYPE :param floor: DESCRIPTION. :type floor: TYPE :return: DESCRIPTION. :rtype: TYPE

use_measurement_error()

Use measurement error.

validate_array_shape(*data*)

Validate array shape. :param *data*: DESCRIPTION. :type *data*: TYPE :return: DESCRIPTION. :rtype: TYPE

validate_percent(*value*)

Make sure the percent is a decimal. :param *value*: DESCRIPTION. :type *value*: TYPE :return: DESCRIPTION. :rtype: TYPE

mtpy.modeling.gocad module

Created on Fri Dec 09 15:50:53 2016

@author: Alison Kirkby

read and write gocad objects

class mtpy.modeling.gocad.Sgrid(kwargs)**

Bases: object

Class to read and write gocad sgrid files

need to provide: workdir = working directory fn = filename for the sgrid resistivity = 3d numpy array containing resistivity values, shape (ny,nx,nz) grid_xyz = tuple containing x,y,z locations of edges of cells for each

resistivity value. Each item in tuple has shape (ny+1,nx+1,nz+1).

read_sgrid_file(headerfn=None)

Read sgrid file.

write_sgrid_file()

Write sgrid file.

mtpy.modeling.mare2dem module

mtpy.modeling.mesh_tools module

Created on Wed Oct 25 09:35:31 2017

@author: Alison Kirkby

functions to assist with mesh generation

mtpy.modeling.mesh_tools.get_nearest_index(*array*, *value*)

Return the index of the nearest value to the provided value in an array:

inputs:

array = array or list of values value = target value.

mtpy.modeling.mesh_tools.get_padding_cells(*cell_width*, *max_distance*, *num_cells*, *stretch*)

Get padding cells, which are exponentially increasing to a given distance. Make sure that each cell is larger than the one previously.

Parameters

- ****cell_width**** – float width of grid cell (m)
- ****max_distance**** – float maximum distance the grid will extend (m)
- ****num_cells**** – int number of padding cells
- ****stretch**** – float base geometric factor

```
mtpy.modeling.mesh_tools.get_padding_cells2(cell_width, core_max, max_distance, num_cells)
```

Get padding cells, which are exponentially increasing to a given distance. Make sure that each cell is larger than the one previously.

```
mtpy.modeling.mesh_tools.get_padding_from_stretch(cell_width, pad_stretch, num_cells)
```

Get padding cells using pad stretch factor.

```
mtpy.modeling.mesh_tools.get_rounding(cell_width)
```

Get the rounding number given the cell width.

Will be one significant number less

than the cell width. This reduces weird looking meshes.

param cell_width

Width of mesh cell.

type cell_width

float

return

Digit to round to.

rtype

int

```
mtpy.modeling.mesh_tools.get_station_buffer(grid_east, grid_north, station_east, station_north,  
buf=10000.0)
```

Get cells within a specified distance (buf) of the stations returns a 2D boolean (True/False) array

```
mtpy.modeling.mesh_tools.grid_centre(grid_edges)
```

Calculate the grid centres from an array that defines grid edges. :param grid_edges: Array containing grid edges.
:return s: Grid_centre: centre points of grid.

```
mtpy.modeling.mesh_tools.interpolate_elevation_to_grid(grid_east, grid_north, utm_epsg=None,  
datum_epsg=4326, surface_file=None,  
surface=None, method='linear', fast=True,  
buffer=1)
```

Note: this documentation is outdated and seems to be copied from # model.interpolate_elevation2. It needs to be updated. This # funciton does not update a dictionary but returns an array of # elevation data.

project a surface to the model grid and add resulting elevation data to a dictionary called surface_dict. Assumes the surface is in lat/long coordinates (wgs84) The ‘fast’ method extracts a subset of the elevation data that falls within the mesh-bounds and interpolates them onto mesh nodes. This approach significantly speeds up (~ x5) the interpolation procedure.

returns nothing returned, but surface data are added to surface_dict under the key given by surfacename.

inputs choose to provide either surface_file (path to file) or surface (tuple). If both are provided then surface tuple takes priority.

surface elevations are positive up, and relative to sea level. surface file format is:

```
ncols 3601 nrows 3601 xllcorner -119.00013888889 (longitude of lower left) yllcorner 36.999861111111 (latitude of lower left) cellsize 0.00027777777777778 NODATA_value -9999 elevation data W -> E N | V S
```

Alternatively, provide a tuple with: (lon,lat,elevation) where elevation is a 2D array (shape (ny,nx)) containing elevation points (order S -> N, W -> E) and lon, lat are either 1D arrays containing list of longitudes and latitudes (in the case of a regular grid) or 2D arrays with same shape as elevation array containing longitude and latitude of each point.

other inputs: surfacename = name of surface for putting into dictionary surface_epsg = epsg number of input surface, default is 4326 for lat/lon(wgs84) method = interpolation method. Default is ‘nearest’, if model grid is dense compared to surface points then choose ‘linear’ or ‘cubic’

```
mtpy.modeling.mesh_tools.make_log_increasing_array(z1_layer, target_depth, n_layers,
increment_factor=0.9)
```

Create depth array with log increasing cells, down to target depth, inputs are z1_layer thickness, target depth, number of layers (n_layers)

```
mtpy.modeling.mesh_tools.rotate_mesh(grid_east, grid_north, origin, rotation_angle, return_centre=False)
```

Rotate a mesh defined by grid_east and grid_north. :param grid_east: 1d array defining the edges of the mesh in the east-west direction. :param grid_north: 1d array defining the edges of the mesh in the north-south direction. :param origin: Real-world position of the (0,0) point in grid_east, grid_north. :param rotation_angle: Angle in degrees to rotate the grid by. :param return_centre: True/False option to return points on centre of grid instead of grid edges, defaults to False. :return: Grid_east, grid_north - 2d arrays describing the east and north coordinates.

[mtpy.modeling.pek1d module](#)

[mtpy.modeling.pek1dclasses module](#)

[mtpy.modeling.pek2d module](#)

[mtpy.modeling.pek2dforward module](#)

[mtpy.modeling.structured_mesh_3d module](#)

ModEM

Generate files for ModEM

revised by JP 2017 # revised by AK 2017 to bring across functionality from ak branch # revised by JP 2021 updating functionality and updating docs

```
class mtpy.modeling.structured_mesh_3d.StructuredGrid3D(station_locations=None,
center_point=None, **kwargs)
```

Bases: object

Make and read a FE mesh grid

The mesh assumes the coordinate system where:

x == North y == East z == + down

All dimensions are in meters.

The mesh is created by first making a regular grid around the station area, then padding cells are added that exponentially increase to the given extensions. Depth cell increase on a log10 scale to the desired depth, then padding cells are added that increase exponentially..

Parameters

station_object	–	mtpy.modeling.modem.Stations	object ..	seealso::
mtpy.modeling.modem.Stations				

Examples

Example 1 → create mesh first then data file

```
>>> import mtpy.modeling.modem as modem
>>> import os
```

(continues on next page)

(continued from previous page)

```
>>> # 1) make a list of all .edi files that will be inverted for
>>> edi_path = r"/home/EDI_Files"
>>> edi_list = [os.path.join(edi_path, edi)
```

```
for edi in os.listdir(edi_path)
```

```
>>> ...           if edi.find('.edi') > 0]
>>> # 2) Make a Stations object
>>> stations_obj = modem.Stations()
>>> stations_obj.get_station_locations_from_edi(edi_list)
>>> # 3) make a grid from the stations themselves with 200m cell_
-> spacing
>>> mmesh = modem.Model(station_obj)
>>> # change cell sizes
>>> mmesh.cell_size_east = 200,
>>> mmesh.cell_size_north = 200
>>> mmesh.ns_ext = 300000 # north-south extension
>>> mmesh.ew_ext = 200000 # east-west extension of model
>>> mmesh.make_mesh()
>>> # check to see if the mesh is what you think it should be
>>> mmsmesh.plot_mesh()
>>> # all is good write the mesh file
>>> mmsmesh.write_model_file(save_path=r"/home/modem/Inv1")
>>> # create data file
>>> md = modem.Data(edi_list, station_locations=mmesh.station_
-> locations)
>>> md.write_data_file(save_path=r"/home/modem/Inv1")
```

Example 2 → Rotate Mesh

```
>>> mmesh.mesh_rotation_angle = 60
>>> mmesh.make_mesh()
```

Note

ModEM assumes all coordinates are relative to North and East, and does not accommodate mesh rotations, therefore, here the rotation is of the stations, which essentially does the same thing. You will need to rotate your data to align with the ‘new’ coordinate system.

Attributes	Description
_logger	python logging object that put messages in logging format defined in logging configure file, see MtPyLog more information
cell_number_ew	optional for user to specify the total number of cells on the east-west direction. <i>default</i> is None
cell_number_ns	optional for user to specify the total number of cells on the north-south direction. <i>default</i> is None
cell_size_east	mesh block width in east direction <i>default</i> is 500

continues on next page

Table 7 – continued from previous page

Attributes	Description
cell_size_north	mesh block width in north direction <i>default</i> is 500
grid_center	center of the mesh grid
grid_east	overall distance of grid nodes in east direction
grid_north	overall distance of grid nodes in north direction
grid_z	overall distance of grid nodes in z direction
model_fn	full path to initial file name
model_fn_basename	default name for the model file name
n_air_layers	number of air layers in the model. <i>default</i> is 0
n_layers	total number of vertical layers in model
nodes_east	relative distance between nodes in east direction
nodes_north	relative distance between nodes in north direction
nodes_z	relative distance between nodes in east direction
pad_east	number of cells for padding on E and W sides <i>default</i> is 7
pad_north	number of cells for padding on S and N sides <i>default</i> is 7
pad_num	number of cells with cell_size with outside of station area. <i>default</i> is 3
pad_method	method to use to create padding: extent1, extent2 - calculate based on ew_ext and ns_ext stretch - calculate based on pad_stretch factors
pad_stretch_h	multiplicative number for padding in horizontal direction.
pad_stretch_v	padding cells N & S will be pad_root_north** <i>(x)</i>
pad_z	number of cells for padding at bottom <i>default</i> is 4
ew_ext	E-W extension of model in meters
ns_ext	N-S extension of model in meters
res_scale	scaling method of res, supports ‘log’ - for log e format ‘log’ or ‘log10’ - for log with base 10 ‘linear’ - linear scale <i>default</i> is ‘log’
res_list	list of resistivity values for starting model
res_model	starting resistivity model
res_initial_value	resistivity initial value for the resistivity model <i>default</i> is 100
mesh_rotation_angle	Angle to rotate the grid to. Angle is measured positive clockwise assuming North is 0 and east is 90. <i>default</i> is None
save_path	path to save file to
sea_level	sea level in grid_z coordinates. <i>default</i> is 0
station_locations	location of stations
title	title in initial file
z1_layer	first layer thickness
z_bottom	absolute bottom of the model <i>default</i> is 300,000
z_target_depth	Depth of deepest target, <i>default</i> is 50,000

add_layers_to_mesh(*n_add_layers=None*, *layer_thickness=None*, *where='top'*)

Function to add constant thickness layers to the top or bottom of mesh.

Note: It is assumed these layers are added before the topography. If you want to add topography layers,

use function add_topography_to_model :param n_add_layers: Integer, number of layers to add, defaults to None. :param layer_thickness: Real value or list/array. Thickness of layers,

Can provide a single value

or a list/array containing multiple layer thicknesses, defaults to None.

Parameters

where – Where to add, top or bottom, defaults to “top”.

```
add_topography_from_data(interp_method='nearest', air_resistivity=1000000000000.0,  
topography_buffer=None, airlayer_type='log_up')
```

Wrapper around add_topography_to_model that allows creating a surface model from EDI data. The Data grid and station elevations will be used to make a ‘surface’ tuple that will be passed to add_topography_to_model so a surface model can be interpolated from it.

The surface tuple is of format (lon, lat, elev) containing station locations. :param data_object: A ModEm data

object that has been filled with data from EDI files.

Parameters

- **interp_method** (*str, optional*) – Same as add_topography_to_model, defaults to “nearest”.
- **air_resistivity** (*float, optional*) – Same as add_topography_to_model, defaults to 1e12.
- **topography_buffer** (*float, optional*) – Same as add_topography_to_model, defaults to None.
- **airlayer_type** (*str, optional*) – Same as add_topography_to_model, defaults to “log_up”.

```
add_topography_to_model(topography_file=None, surface=None, topography_array=None,  
interp_method='nearest', air_resistivity=1000000000000.0,  
topography_buffer=None, airlayer_type='log_up', max_elev=None,  
shift_east=0, shift_north=0)
```

If air_layers is non-zero, will add topo: read in topograph file, make a surface model.

Call project_stations_on_topography in the end, which will re-write the .dat file.

If n_airlayers is zero, then cannot add topo data, only bathymetry is needed. :param shift_north:

Defaults to 0.

Parameters

- **shift_east** – Defaults to 0.
- **max_elev** – Defaults to None.
- **surface** – Defaults to None.
- **topography_file** – File containing topography (arcgis ascii grid), defaults to None.
- **topography_array** – Alternative to topography_file - array of elevation values on model grid, defaults to None.
- **interp_method** – Interpolation method for topography, ‘nearest’, ‘linear’, or ‘cubic’, defaults to “nearest”.

- **air_resistivity** – Resistivity value to assign to air, defaults to 1e12.
- **topography_buffer** – Buffer around stations to calculate minimum and maximum topography value to use for meshing, defaults to None.
- **airlayer_type** – How to set air layer thickness - options are ‘constant’ for constant air layer thickness, or ‘log’, for logarithmically increasing air layer thickness upward, defaults to “log_up”.

assign_resistivity_from_surface_data(*top_surface, bottom_surface, resistivity_value*)

Assign resistivity value to all points above or below a surface requires the surface_dict attribute to exist and contain data for surface key (can get this information from ascii file using project_surface)

inputs surface_name = name of surface (must correspond to key in surface_dict) resistivity_value = value to assign where = ‘above’ or ‘below’ - assign resistivity above or below the

surface

convert_model_to_int(*res_list=None*)

Convert resistivity values to integers according to resistivity list. :param res_list: Resistivity values in Ohm-m, defaults to None. :type res_list: list of floats, optional :return: Array of integers corresponding to the res_list. :rtype: np.ndarray(dtype=int)

estimate_skin_depth(*apparent_resistivity, period, scale='km'*)

Estimate skin depth from apparent resistivity and period. :param apparent_resistivity: DESCRIPTION. :type apparent_resistivity: TYPE :param period: DESCRIPTION. :type period: TYPE :param scale: DESCRIPTION, defaults to “km”. :type scale: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

from_gocad_sgrid(*sgrid_header_file, air_resistivity=1e+39, sea_resistivity=0.3, sgrid_positive_up=True*)

Read a gocad sgrid file and put this info into a ModEM file.

Note: can only deal with grids oriented N-S or E-W at this stage, with orthogonal coordinates

from_modem(*model_fn=None*)

Read an initial file and return the pertinent information including grid positions in coordinates relative to the center point (0,0) and starting model.

Note that the way the model file is output, it seems is that the blocks are setup as

ModEM: WS::

0——> N_north 0——>N_east ||| V V N_east N_north

Arguments:

```
**model_fn** : full path to initializing file.
```

Outputs:

```
**nodes_north** : np.array(nx)
array of nodes in S --> N direction

**nodes_east** : np.array(ny)
array of nodes in the W --> E direction

**nodes_z** : np.array(nz)
array of nodes in vertical direction positive downwards

**res_model** : dictionary
```

(continues on next page)

(continued from previous page)

```
    dictionary of the starting model with keys as layers

**res_list** : list
    list of resistivity values in the model

**title** : string
    title string
```

from_ws3dinv(model_fn)

Read WS3DINV iteration model file. :param model_fn: DESCRIPTION. :type model_fn: TYPE :return: DESCRIPTION. :rtype: TYPE

from_ws3dinv_initial(initial_fn)

Read an initial file and return the pertinent information including grid positions in coordinates relative to the center point (0,0) and starting model.

Arguments:

```
**initial_fn** : full path to initializing file.
```

Outputs:

```
**nodes_north** : np.array(nx)
    array of nodes in S --> N direction

**nodes_east** : np.array(ny)
    array of nodes in the W --> E direction

**nodes_z** : np.array(nz)
    array of nodes in vertical direction positive downwards

**res_model** : dictionary
    dictionary of the starting model with keys as layers

**res_list** : list
    list of resistivity values in the model

**title** : string
    title string
```

get_lower_left_corner(pad_east, pad_north, shift_east=0, shift_north=0)

Get the lower left corner in UTM coordinates for raster. :param shift_north:

Defaults to 0.

Parameters

- **shift_east** – Defaults to 0.
- **pad_east** (*integer*) – Number of padding cells to skip from outside in.
- **pad_north** (*integer*) – Number of padding cells to skip from outside in.

Returns

Lower left hand corner.

Return type`mtpy.core.MTLocation`

interpolate_elevation(*surface_file=None*, *surface=None*, *get_surface_name=False*, *method='nearest'*, *fast=True*, *shift_north=0*, *shift_east=0*)

Project a surface to the model grid and add resulting elevation data to a dictionary called *surface_dict*. Assumes the surface is in lat/long coordinates (wgs84)

returns nothing returned, but surface data are added to *surface_dict* under the key given by *surface_name*.

inputs choose to provide either *surface_file* (path to file) or *surface* (tuple). If both are provided then *surface tuple* takes priority.

surface elevations are positive up, and relative to sea level. surface file format is:

```
ncols 3601 nrows 3601 xllcorner -119.00013888889 (longitude of lower left) yllcorner 36.999861111111  
(latitude of lower left) cellsize 0.0002777777777778 NODATA_value -9999 elevation data W -> E N |  
V S
```

Alternatively, provide a tuple with: (lon,lat,elevation) where elevation is a 2D array (shape (ny,nx)) containing elevation points (order S -> N, W -> E) and lon, lat are either 1D arrays containing list of longitudes and latitudes (in the case of a regular grid) or 2D arrays with same shape as elevation array containing longitude and latitude of each point.

other inputs: *surface_epsg* = epsg number of input surface, default is 4326 for lat/lon(wgs84) *method* = interpolation method. Default is ‘nearest’, if model grid is dense compared to surface points then choose ‘linear’ or ‘cubic’

interpolate_to_even_grid(*cell_size*, *pad_north=None*, *pad_east=None*)

Interpolate the model onto an even grid for plotting as a raster or netCDF. :param *cell_size*: DESCRIPTION. :type *cell_size*: TYPE :param *pad_north*: DESCRIPTION, defaults to None. :type *pad_north*: TYPE, optional :param *pad_east*: DESCRIPTION, defaults to None. :type *pad_east*: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

make_mesh(*verbose=True*)

Create finite element mesh according to user-input parameters.

The mesh is built by:

1. Making a regular grid within the station area.
 - Uses *cell_size_east* and *cell_size_north* for dimensions
2. Adding *pad_num* of *cell_width* cells outside of station area
3. Adding padding cells to given extension and number of padding cells. - *extent1* - stretch to a given distance with *pad_east* or *pad_north* number of cells.
 - *extent2* - stretch to a given distance with *pad_east* or *pad_north* number of cells.
 - *stretch* stretches from station area using *pad_north* and *pad_east* times *pad_stretch_h*

4. Making vertical cells starting with `z1_layer` increasing logarithmically (base 10) to `z_target_depth` and `num_layers`. - *default* creates a vertical mesh that increases logarithmically down. See `make_z_mesh`.
 - *custom* input your own vertical mesh.
5. Add vertical padding cells to desired extension.
6. Check to make sure none of the stations lie on a node. If they do then move the node by `.02*cell_width`

`make_z_mesh(n_layers=None)`

New version of `make_z_mesh`. `make_z_mesh` and `M`.

`property model_epsg`

Model epsg.

`property model_fn`

Model fn.

`property model_parameters`

Get important model parameters to write to a file for documentation later.

`property nodes_east`

Nodes east.

`property nodes_north`

Nodes north.

`property nodes_z`

Nodes z.

`property plot_east`

Plot east.

`plot_mesh(kwargs)`**

Plot model mesh. :param `**kwargs`: :param `plot_topography`: DESCRIPTION, defaults to False. :type `plot_topography`: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

`property plot_north`

Plot north.

`property plot_z`

Plot z.

`property save_path`

Save path.

`to_conductance_raster(cell_size, conductance_dict, pad_north=None, pad_east=None, save_path=None, rotation_angle=0, shift_north=0, shift_east=0, log10=True, verbose=True)`

Write out a raster in UTM coordinates for conductance sections.

Expecting a grid that is interoplated onto a regular grid of square cells with size `cell_size`.

```conductance\_dict = {"layer\_name": (min\_depth, max\_depth)}```

if “layer\_name” is “surface” then topography is included :param verbose:

Defaults to True.

**Parameters**

- **log10** – Defaults to True.
- **shift\_east** – Defaults to 0.
- **shift\_north** – Defaults to 0.
- **cell\_size** (*float*) – Square cell size (cell\_size x cell\_size) in meters.
- **conductance\_dict** (*TYPE*) – DESCRIPTION.
- **pad\_north** (*integer, optional*) – Number of padding cells to skip from outside in, if None defaults to self.pad\_north, defaults to None.
- **pad\_east** (*integer, optional*) – Number of padding cells to skip from outside in if None defaults to self.pad\_east, defaults to None.
- **save\_path** (*string or Path, optional*) – Path to save files to. If None use self.save\_path, defaults to None.
- **depth\_min** (*float, optional*) – Minimum depth to make raster for in meters,, defaults to None which will use shallowest depth.
- **depth\_max** (*float, optional*) – Maximum depth to make raster for in meters,, defaults to None which will use deepest depth.
- **rotation\_angle** (*float, optional*) – Angle (degrees) to rotate the raster assuming clockwise positive rotation where North = 0, East = 90, defaults to 0.

**Raises**

**ValueError** – If utm\_epsg is not input.

**Returns**

List of file paths to rasters.

**Return type**

*TYPE*

**to\_geosoft\_xyz(*save\_fn, pad\_north=0, pad\_east=0, pad\_z=0*)**

Write an XYZ file readable by Geosoft

All input units are in meters.. :param save\_fn: Full path to save file to. :type save\_fn: string or Path :param pad\_north: Number of cells to cut from the north-south edges, defaults to 0. :type pad\_north: int, optional :param pad\_east: Number of cells to cut from the east-west edges, defaults to 0. :type pad\_east: int, optional :param pad\_z: Number of cells to cut from the bottom, defaults to 0. :type pad\_z: int, optional

**to\_gocad\_sgrid(*fn=None, origin=[0, 0, 0], clip=0, no\_data\_value=-99999*)**

Write a model to gocad sgrid

optional inputs:

**fn = filename to save to. File extension ('.sg') will be appended.**

default is the model name with extension removed

origin = real world [x,y,z] location of zero point in model grid clip = how much padding to clip off the edge of the model for export,

provide one integer value or list of 3 integers for x,y,z directions

no\_data\_value = no data value to put in sgrid.

**to\_modem**(model\_fn=None, \*\*kwargs)

Will write an initial file for ModEM.

Note that x is assumed to be S → N, y is assumed to be W → E and z is positive downwards. This means that index [0, 0, 0] is the southwest corner of the first layer. Therefore if you build a model by hand the layer block will look as it should in map view.

Also, the xgrid, ygrid and zgrid are assumed to be the relative distance between neighboring nodes. This is needed because wsinv3d builds the model from the bottom SW corner assuming the cell width from the init file.

Key Word Arguments:

```
model_fn_basename : string
 basename to save file to
 default is ModEM_Model.ws
 file is saved at save_path/model_fn_basename

title : string
 Title that goes into the first line
 default is Model File written by MTpy.modeling.modem

res_starting_value : float
 starting model resistivity value,
 assumes a half space in Ohm-m
 default is 100 Ohm-m

res_scale : ['log' | 'log10' | 'linear']
 scale of resistivity. In the ModEM code it
 converts everything to Loge,
 default is 'log'
```

**to\_netcdf**(fn, pad\_east=None, pad\_north=None, metadata={})

Create a netCDF file to read into GIS software

works about 50% of the time..

**to\_raster**(cell\_size, pad\_north=None, pad\_east=None, save\_path=None, depth\_min=None, depth\_max=None, rotation\_angle=0, shift\_north=0, shift\_east=0, log10=True, verbose=True)

Write out each depth slice as a raster in UTM coordinates.

Expecting

**a grid that is interpolated onto a regular grid of square cells with size cell\_size.** :param verbose:

Defaults to True.

**param log10**

Defaults to True.

**param shift\_east**

Defaults to 0.

**param shift\_north**

Defaults to 0.

---

**param cell\_size**  
Square cell size (cell\_size x cell\_size) in meters.

**type cell\_size**  
float

**param pad\_north**  
Number of padding cells to skip from outside in, if None defaults to self.pad\_north, defaults to None.

**type pad\_north**  
integer, optional

**param pad\_east**  
Number of padding cells to skip from outside in if None defaults to self.pad\_east, defaults to None.

**type pad\_east**  
integer, optional

**param save\_path**  
Path to save files to. If None use self.save\_path, defaults to None.

**type save\_path**  
string or Path, optional

**param depth\_min**  
Minimum depth to make raster for in meters,, defaults to None.

**type depth\_min**  
float, optional

**param depth\_max**  
Maximum depth to make raster for in meters,, defaults to None.

**type depth\_max**  
float, optional

**param rotation\_angle**  
Angle (degrees) to rotate the raster assuming clockwise positive rotation where North = 0, East = 90, defaults to 0.

**type rotation\_angle**  
float, optional

**raises ValueError**  
If utm\_epsg is not input.

**return**  
List of file paths to rasters.

**rtype**  
TYPE

**to\_ubc(basename)**  
Write a UBC .msh and .mod file. :param basename: :param save\_fn: DESCRIPTION. :type save\_fn: TYPE :return: DESCRIPTION. :rtype: TYPE

**to\_vtk(vtk\_fn=None, vtk\_save\_path=None, vtk\_fn\_basename='ModEM\_model\_res', \*\*kwargs)**  
Write a VTK file to plot in 3D rendering programs like Paraview. :param \*\*kwargs: :param vtk\_fn: Full path to VKT file to be written, defaults to None. :type vtk\_fn: string or Path, optional :param

vtk\_save\_path: Directory to save vtk file to, defaults to None. :type vtk\_save\_path: string or Path, optional :param vtk\_fn\_basename: Filename basename of vtk file, note that .vtr extension is automatically added, defaults to “ModEM\_model\_res”.

#### Parameters

- **geographic\_coordinates** (*boolean, optional*) – [ True | False ] True for geographic coordinates.
- **units** (*string, optional*) – Units of the spatial grid [ km | m | ft ], defaults to “km”.
- **coordinate\_system** – Coordinate system for the station, either the normal MT right-hand coordinate system with z+ down or the sinister z- down [ nez+ | enz- ], defaults to nez+.

#### Type

string

#### Returns

Full path to VTK file.

#### Return type

Path

### **to\_winglink\_out**(*save\_fn*)

Will write an .out file for LeapFrog.

Note that y is assumed to be S → N, e is assumed to be W → E and z is positive upwards. This means that index [0, 0, 0] is the southwest corner of the first layer. :param save\_fn: Full path to save file to. :type save\_fn: string or Path :return: DESCRIPTION. :rtype: TYPE

### **to\_ws3dinv\_intial**(*initial\_fn, res\_list=None*)

Write a WS3DINV intial model file.

### **to\_xarray**(\*\**kwargs*)

Put model in xarray format.

### **to\_xyres**(*save\_path=None, location\_type='EN', depth\_index='all', outfile\_basename='DepthSlice', log\_res=False, pad\_east=None, pad\_north=None*)

Write files containing depth slice data (x, y, res for each depth)

#### **origin = x,y coordinate of zero point of ModEM\_grid, or name of file**

containing this info (full path or relative to model files)

save\_path = path to save to, default is the model object save path location\_type = ‘EN’ or ‘LL’ xy points saved as eastings/northings or

longitude/latitude, if ‘LL’ need to also provide model\_epsg

model\_epsg = epsg number that was used to project the model outfile\_basename = string for basename for saving the depth slices. log\_res = True/False - option to save resistivity values as log10

instead of linear

clip = number of cells to clip on each of the east/west and north/south edges.

### **to\_xyzres**(*savefile=None, location\_type='EN', log\_res=False, pad\_east=None, pad\_north=None*)

Save a model file as a space delimited x y z res file.

**mtpy.modeling.winglink module**

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)**

Bases: object

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
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

continues on next page

Table 8 – continued from previous page

key words	description
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

Meth-ods	Description
plot	plots a pseudo-section of apparent resistivity 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_fig()	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.

**Arguments:****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)**

Bases: object

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

Meth- ods	Description
plot	plots a pseudo-section of apparent resistivity 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_figur	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"
```

(continues on next page)

(continued from previous page)

```
>>> 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.

**Arguments:****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*)

Bases: object

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

**Arguments:****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 → ma
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
mtd	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

Attributes/key words	description
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
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-	call redraw_plot to redraw the figures, if one of the attributes has been changed
draw_plot	
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.Occam2DDData(r"/home/occam2d/Data.dat")
>>> p1 = ocd.plot2DResponses()
>>> p1.save_figures(r"/home/occam2d/Figures", fig_fmt='jpg')
```

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

Bases: Exception

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.

#### Arguments:

##### output\_fn

[string] the full path to winglink outputfile

#### Returns:

##### 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

## mtpy.modeling.winglinktools module

Created on Mon Aug 22 15:19:30 2011

@author: a1185872

mtpy.modeling.winglinktools.plotResponses(outputfile, maxcol=8, plottype='all', \*\*kwargs)

PlotResponse will plot the responses modeled from winglink against the observed data.

#### Inputs:

outputfile = full path and filename to output file maxcol = maximum number of columns for the plot  
plottype = ‘all’ to plot all on the same plot

‘1’ to plot each responses in a different figure station to plot a single station or enter as a list of stations to plot a few stations [station1,station2]. Does not have to be verbatim but should have similar unique characters input pb01 for pb01cs in outputfile

**Outputs:**

None

```
mtpy.modeling.winglinktools.readModelFile(modelfile, profiledirection='ew')
```

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.winglinktools.readOutputFile(outputfile)
```

ReadOutputFile will read an output file from winglink and output data in the form of a dictionary.

**Input:**

outputfile = the full path and filename of outputfile

**Output:****idict = dictionary with keys of station name**

each idict[station name] is a dictionary with keys corresponding to modeled and observed responses:

‘obsresxy’, ‘obsphasexy’, ‘modresxy’, ‘modphasexy’, ‘obsresyx’, ‘ob-  
phaseyx’, ‘modresyx’, ‘modphasexy’, ‘obshzres’, ‘obshzphase’, ‘modhzres’, ‘modhzphase’, ‘period’

**rplst = list of dictionaries for each station with keywords:**

‘station’ = station name ‘offset’ = relative offset, ‘resxy’ = TE resistivity and error as row 0 and 1 respectively, ‘resyx’ = TM resistivity and error as row 0 and 1 respectively, ‘phasexy’ = TE phase and error as row 0 and 1 respectively, ‘phaseyx’ = Tm phase and error as row 0 and 1 respectively, ‘realtip’ = Real Tipper and error as row 0 and 1 respectively, ‘imagtip’ = Imaginary Tipper and error as row 0 and 1 respectively

plst = periodlst as the median of all stations. stationlst = list of stations in order from profile title = list of parameters for plotting as [title,profile,inversiontype]

## Module contents

```
class mtpy.modeling.StructuredGrid3D(station_locations=None, center_point=None, **kwargs)
```

Bases: object

Make and read a FE mesh grid

**The mesh assumes the coordinate system where:**

x == North y == East z == + down

All dimensions are in meters.

The mesh is created by first making a regular grid around the station area, then padding cells are added that exponentially increase to the given extensions. Depth cell increase on a log10 scale to the desired depth, then padding cells are added that increase exponentially..

**Parameters**

**\*\*station\_object\*\*** – mtpy.modeling.modem.Stations object .. seealso::  
mtpy.modeling.modem.Stations

## Examples

**Example 1 → create mesh first then data file**

```

>>> import mtpy.modeling.modem as modem
>>> import os
>>> # 1) make a list of all .edi files that will be inverted for
>>> edi_path = r"/home/EDI_Files"
>>> edi_list = [os.path.join(edi_path, edi)

for edi in os.listdir(edi_path)

>>> ... if edi.find('.edi') > 0]
>>> # 2) Make a Stations object
>>> stations_obj = modem.Stations()
>>> stations_obj.get_station_locations_from_edi(edi_list)
>>> # 3) make a grid from the stations themselves with 200m cell_
->spacing
>>> mmesh = modem.Model(station_obj)
>>> # change cell sizes
>>> mmesh.cell_size_east = 200,
>>> mmesh.cell_size_north = 200
>>> mmesh.ns_ext = 300000 # north-south extension
>>> mmesh.ew_ext = 200000 # east-west extension of model
>>> mmesh.make_mesh()
>>> # check to see if the mesh is what you think it should be
>>> mmsmesh.plot_mesh()
>>> # all is good write the mesh file
>>> mmsmesh.write_model_file(save_path=r"/home/modem/Inv1")
>>> # create data file
>>> md = modem.Data(edi_list, station_locations=mmesh.station_
->locations)
>>> md.write_data_file(save_path=r"/home/modem/Inv1")

```

### Example 2 → Rotate Mesh

```

>>> mmesh.mesh_rotation_angle = 60
>>> mmesh.make_mesh()

```

#### Note

ModEM assumes all coordinates are relative to North and East, and does not accommodate mesh rotations, therefore, here the rotation is of the stations, which essentially does the same thing. You will need to rotate your data to align with the ‘new’ coordinate system.

Attributes	Description
_logger	python logging object that put messages in logging format defined in logging configure file, see MtPyLog more information
cell_number_ew	optional for user to specify the total number of cells on the east-west direction. <i>default</i> is None

continues on next page

Table 11 – continued from previous page

Attributes	Description
cell_number_ns	optional for user to specify the total number of sells on the north-south direction. <i>default</i> is None
cell_size_east	mesh block width in east direction <i>default</i> is 500
cell_size_north	mesh block width in north direction <i>default</i> is 500
grid_center	center of the mesh grid
grid_east	overall distance of grid nodes in east direction
grid_north	overall distance of grid nodes in north direction
grid_z	overall distance of grid nodes in z direction
model_fn	full path to initial file name
model_fn_basename	default name for the model file name
n_air_layers	number of air layers in the model. <i>default</i> is 0
n_layers	total number of vertical layers in model
nodes_east	relative distance between nodes in east direction
nodes_north	relative distance between nodes in north direction
nodes_z	relative distance between nodes in east direction
pad_east	number of cells for padding on E and W sides <i>default</i> is 7
pad_north	number of cells for padding on S and N sides <i>default</i> is 7
pad_num	number of cells with cell_size with outside of station area. <i>default</i> is 3
pad_method	method to use to create padding: extent1, extent2 - calculate based on ew_ext and ns_ext stretch - calculate based on pad_stretch factors
pad_stretch_h	multiplicative number for padding in horizontal direction.
pad_stretch_v	padding cells N & S will be pad_root_north** <i>(x)</i>
pad_z	number of cells for padding at bottom <i>default</i> is 4
ew_ext	E-W extension of model in meters
ns_ext	N-S extension of model in meters
res_scale	<b>scaling method of res, supports</b> ‘log’ - for log e format ‘log’ or ‘log10’ - for log with base 10 ‘linear’ - linear scale <i>default</i> is ‘log’
res_list	list of resistivity values for starting model
res_model	starting resistivity model
res_initial_value	resistivity initial value for the resistivity model <i>default</i> is 100
mesh_rotation_angle	Angle to rotate the grid to. Angle is measured positive clockwise assuming North is 0 and east is 90. <i>default</i> is None
save_path	path to save file to
sea_level	sea level in grid_z coordinates. <i>default</i> is 0
station_locations	location of stations
title	title in initial file
z1_layer	first layer thickness
z_bottom	absolute bottom of the model <i>default</i> is 300,000
z_target_depth	Depth of deepest target, <i>default</i> is 50,000

`add_layers_to_mesh(n_add_layers=None, layer_thickness=None, where='top')`

Function to add constant thickness layers to the top or bottom of mesh.

Note: It is assumed these layers are added before the topography. If you want to add topography layers, use function `add_topography_to_model` :param `n_add_layers`: Integer, number of layers to add, defaults to None. :param `layer_thickness`: Real value or list/array. Thickness of layers,

Can provide a single value

or a list/array containing multiple layer thicknesses, defaults to None.

#### Parameters

**where** – Where to add, top or bottom, defaults to “top”.

```
add_topography_from_data(interp_method='nearest', air_resistivity=1000000000000.0,
topography_buffer=None, airlayer_type='log_up')
```

Wrapper around `add_topography_to_model` that allows creating a surface model from EDI data. The Data grid and station elevations will be used to make a ‘surface’ tuple that will be passed to `add_topography_to_model` so a surface model can be interpolated from it.

The surface tuple is of format (lon, lat, elev) containing station locations. :param `data_object`: A ModEm data

object that has been filled with data from EDI files.

#### Parameters

- **interp\_method** (*str, optional*) – Same as `add_topography_to_model`, defaults to “nearest”.
- **air\_resistivity** (*float, optional*) – Same as `add_topography_to_model`, defaults to 1e12.
- **topography\_buffer** (*float, optional*) – Same as `add_topography_to_model`, defaults to None.
- **airlayer\_type** (*str, optional*) – Same as `add_topography_to_model`, defaults to “log\_up”.

```
add_topography_to_model(topography_file=None, surface=None, topography_array=None,
interp_method='nearest', air_resistivity=1000000000000.0,
topography_buffer=None, airlayer_type='log_up', max_elev=None,
shift_east=0, shift_north=0)
```

If `air_layers` is non-zero, will add topo: read in topograph file, make a surface model.

Call `project_stations_on_topography` in the end, which will re-write the .dat file.

If `n_airlayers` is zero, then cannot add topo data, only bathymetry is needed. :param `shift_north`:

Defaults to 0.

#### Parameters

- **shift\_east** – Defaults to 0.
- **max\_elev** – Defaults to None.
- **surface** – Defaults to None.
- **topography\_file** – File containing topography (arcgis ascii grid), defaults to None.
- **topography\_array** – Alternative to `topography_file` - array of elevation values on model grid, defaults to None.

- **interp\_method** – Interpolation method for topography, ‘nearest’, ‘linear’, or ‘cubic’, defaults to “nearest”.
- **air\_resistivity** – Resistivity value to assign to air, defaults to 1e12.
- **topography\_buffer** – Buffer around stations to calculate minimum and maximum topography value to use for meshing, defaults to None.
- **airlayer\_type** – How to set air layer thickness - options are ‘constant’ for constant air layer thickness, or ‘log’, for logarithmically increasing air layer thickness upward, defaults to “log\_up”.

**assign\_resistivity\_from\_surface\_data**(*top\_surface*, *bottom\_surface*, *resistivity\_value*)

Assign resistivity value to all points above or below a surface requires the surface\_dict attribute to exist and contain data for surface key (can get this information from ascii file using project\_surface)

**inputs** *surface\_name* = name of surface (must correspond to key in surface\_dict) *resistivity\_value* = value to assign where = ‘above’ or ‘below’ - assign resistivity above or below the

surface

**convert\_model\_to\_int**(*res\_list=None*)

Convert resistivity values to integers according to resistivity list. :param res\_list: Resistivity values in Ohm-m, defaults to None. :type res\_list: list of floats, optional :return: Array of integers corresponding to the res\_list. :rtype: np.ndarray(dtype=int)

**estimate\_skin\_depth**(*apparent\_resistivity*, *period*, *scale='km'*)

Estimate skin depth from apparent resistivity and period. :param apparent\_resistivity: DESCRIPTION. :type apparent\_resistivity: TYPE :param period: DESCRIPTION. :type period: TYPE :param scale: DESCRIPTION, defaults to “km”. :type scale: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

**from\_gocad\_sgrid**(*sgrid\_header\_file*, *air\_resistivity=1e+39*, *sea\_resistivity=0.3*, *sgrid\_positive\_up=True*)

Read a gocad sgrid file and put this info into a ModEM file.

Note: can only deal with grids oriented N-S or E-W at this stage, with orthogonal coordinates

**from\_modem**(*model\_fn=None*)

Read an initial file and return the pertinent information including grid positions in coordinates relative to the center point (0,0) and starting model.

Note that the way the model file is output, it seems is that the blocks are setup as

**ModEM: WS::**

0——> N\_north 0——>N\_east ||| V V N\_east N\_north

Arguments:

```
model_fn : full path to initializing file.
```

Outputs:

```
nodes_north : np.array(nx)
 array of nodes in S --> N direction

nodes_east : np.array(ny)
 array of nodes in the W --> E direction

nodes_z : np.array(nz)
 array of nodes in vertical direction positive downwards
```

(continues on next page)

(continued from previous page)

```
res_model : dictionary
 dictionary of the starting model with keys as layers

res_list : list
 list of resistivity values in the model

title : string
 title string
```

**from\_ws3dinv(model\_fn)**

Read WS3DINV iteration model file. :param model\_fn: DESCRIPTION. :type model\_fn: TYPE :return: DESCRIPTION. :rtype: TYPE

**from\_ws3dinv\_initial(initial\_fn)**

Read an initial file and return the pertinent information including grid positions in coordinates relative to the center point (0,0) and starting model.

Arguments:

```
initial_fn : full path to initializing file.
```

Outputs:

```
nodes_north : np.array(nx)
 array of nodes in S --> N direction

nodes_east : np.array(ny)
 array of nodes in the W --> E direction

nodes_z : np.array(nz)
 array of nodes in vertical direction positive downwards

res_model : dictionary
 dictionary of the starting model with keys as layers

res_list : list
 list of resistivity values in the model

title : string
 title string
```

**get\_lower\_left\_corner(pad\_east, pad\_north, shift\_east=0, shift\_north=0)**

Get the lower left corner in UTM coordinates for raster. :param shift\_north:

Defaults to 0.

**Parameters**

- **shift\_east** – Defaults to 0.
- **pad\_east** (*integer*) – Number of padding cells to skip from outside in.
- **pad\_north** (*integer*) – Number of padding cells to skip from outside in.

**Returns**

Lower left hand corner.

**Return type**

`mtpy.core.MTLocation`

**interpolate\_elevation**(*surface\_file=None*, *surface=None*, *get\_surface\_name=False*, *method='nearest'*, *fast=True*, *shift\_north=0*, *shift\_east=0*)

Project a surface to the model grid and add resulting elevation data to a dictionary called *surface\_dict*. Assumes the surface is in lat/long coordinates (wgs84)

**returns** nothing returned, but surface data are added to *surface\_dict* under the key given by *surface\_name*.

**inputs** choose to provide either *surface\_file* (path to file) or *surface* (tuple). If both are provided then *surface tuple* takes priority.

surface elevations are positive up, and relative to sea level. surface file format is:

```
ncols 3601 nrows 3601 xllcorner -119.00013888889 (longitude of lower left) yllcorner 36.999861111111
(latitude of lower left) cellsize 0.0002777777777778 NODATA_value -9999 elevation data W -> E N |
VS
```

Alternatively, provide a tuple with: (lon,lat,elevation) where elevation is a 2D array (shape (ny,nx)) containing elevation points (order S -> N, W -> E) and lon, lat are either 1D arrays containing list of longitudes and latitudes (in the case of a regular grid) or 2D arrays with same shape as elevation array containing longitude and latitude of each point.

other inputs: *surface\_epsg* = epsg number of input surface, default is 4326 for lat/lon(wgs84) *method* = interpolation method. Default is ‘nearest’, if model grid is dense compared to surface points then choose ‘linear’ or ‘cubic’

**interpolate\_to\_even\_grid**(*cell\_size*, *pad\_north=None*, *pad\_east=None*)

Interpolate the model onto an even grid for plotting as a raster or netCDF. :param *cell\_size*: DESCRIPTION. :type *cell\_size*: TYPE :param *pad\_north*: DESCRIPTION, defaults to None. :type *pad\_north*: TYPE, optional :param *pad\_east*: DESCRIPTION, defaults to None. :type *pad\_east*: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

**make\_mesh**(*verbose=True*)

Create finite element mesh according to user-input parameters.

The mesh is built by:

1. Making a regular grid within the station area.
  - Uses *cell\_size\_east* and *cell\_size\_north* for dimensions
2. Adding *pad\_num* of cell\_width cells outside of station area
3. Adding padding cells to given extension and number of padding cells. - *extent1* - stretch to a given distance with *pad\_east* or *pad\_north* number of cells.
  - *extent2* - stretch to a given distance with *pad\_east* or *pad\_north* number of cells.
  - *stretch* stretches from station area using *pad\_north* and *pad\_east* times *pad\_stretch\_h*

4. Making vertical cells starting with `z1_layer` increasing logarithmically (base 10) to `z_target_depth` and `num_layers`. - *default* creates a vertical mesh that increases logarithmically down. See `make_z_mesh`.
  - *custom* input your own vertical mesh.
5. Add vertical padding cells to desired extension.
6. Check to make sure none of the stations lie on a node. If they do then move the node by `.02*cell_width`

**`make_z_mesh(n_layers=None)`**

New version of `make_z_mesh`. `make_z_mesh` and `M`.

**`property model_epsg`**

Model epsg.

**`property model_fn`**

Model fn.

**`property model_parameters`**

Get important model parameters to write to a file for documentation later.

**`property nodes_east`**

Nodes east.

**`property nodes_north`**

Nodes north.

**`property nodes_z`**

Nodes z.

**`property plot_east`**

Plot east.

**`plot_mesh(**kwargs)`**

Plot model mesh. :param `**kwargs`: :param `plot_topography`: DESCRIPTION, defaults to False. :type `plot_topography`: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

**`property plot_north`**

Plot north.

**`property plot_z`**

Plot z.

**`property save_path`**

Save path.

**`to_conductance_raster(cell_size, conductance_dict, pad_north=None, pad_east=None, save_path=None, rotation_angle=0, shift_north=0, shift_east=0, log10=True, verbose=True)`**

Write out a raster in UTM coordinates for conductance sections.

Expecting a grid that is interoplated onto a regular grid of square cells with size `cell_size`.

`conductance\_dict = {"layer\_name": (min\_depth, max\_depth)}

if "layer\_name" is "surface" then topography is included :param verbose:

Defaults to True.

**Parameters**

- **log10** – Defaults to True.
- **shift\_east** – Defaults to 0.
- **shift\_north** – Defaults to 0.
- **cell\_size** (*float*) – Square cell size (cell\_size x cell\_size) in meters.
- **conductance\_dict** (*TYPE*) – DESCRIPTION.
- **pad\_north** (*integer, optional*) – Number of padding cells to skip from outside in, if None defaults to self.pad\_north, defaults to None.
- **pad\_east** (*integer, optional*) – Number of padding cells to skip from outside in if None defaults to self.pad\_east, defaults to None.
- **save\_path** (*string or Path, optional*) – Path to save files to. If None use self.save\_path, defaults to None.
- **depth\_min** (*float, optional*) – Minimum depth to make raster for in meters,, defaults to None which will use shallowest depth.
- **depth\_max** (*float, optional*) – Maximum depth to make raster for in meters,, defaults to None which will use deepest depth.
- **rotation\_angle** (*float, optional*) – Angle (degrees) to rotate the raster assuming clockwise positive rotation where North = 0, East = 90, defaults to 0.

**Raises**

**ValueError** – If utm\_epsg is not input.

**Returns**

List of file paths to rasters.

**Return type**

*TYPE*

**to\_geosoft\_xyz(*save\_fn, pad\_north=0, pad\_east=0, pad\_z=0*)**

Write an XYZ file readable by Geosoft

All input units are in meters.. :param save\_fn: Full path to save file to. :type save\_fn: string or Path :param pad\_north: Number of cells to cut from the north-south edges, defaults to 0. :type pad\_north: int, optional :param pad\_east: Number of cells to cut from the east-west edges, defaults to 0. :type pad\_east: int, optional :param pad\_z: Number of cells to cut from the bottom, defaults to 0. :type pad\_z: int, optional

**to\_gocad\_sgrid(*fn=None, origin=[0, 0, 0], clip=0, no\_data\_value=-99999*)**

Write a model to gocad sgrid

optional inputs:

**fn = filename to save to. File extension ('.sg') will be appended.**

default is the model name with extension removed

origin = real world [x,y,z] location of zero point in model grid clip = how much padding to clip off the edge of the model for export,

provide one integer value or list of 3 integers for x,y,z directions

no\_data\_value = no data value to put in sgrid.

**to\_modem**(model\_fn=None, \*\*kwargs)

Will write an initial file for ModEM.

Note that x is assumed to be S → N, y is assumed to be W → E and z is positive downwards. This means that index [0, 0, 0] is the southwest corner of the first layer. Therefore if you build a model by hand the layer block will look as it should in map view.

Also, the xgrid, ygrid and zgrid are assumed to be the relative distance between neighboring nodes. This is needed because wsinv3d builds the model from the bottom SW corner assuming the cell width from the init file.

Key Word Arguments:

```
model_fn_basename : string
 basename to save file to
 default is ModEM_Model.ws
 file is saved at save_path/model_fn_basename

title : string
 Title that goes into the first line
 default is Model File written by MTpy.modeling.modem

res_starting_value : float
 starting model resistivity value,
 assumes a half space in Ohm-m
 default is 100 Ohm-m

res_scale : ['log' | 'log10' | 'linear']
 scale of resistivity. In the ModEM code it
 converts everything to Loge,
 default is 'log'
```

**to\_netcdf**(fn, pad\_east=None, pad\_north=None, metadata={})

Create a netCDF file to read into GIS software

works about 50% of the time..

**to\_raster**(cell\_size, pad\_north=None, pad\_east=None, save\_path=None, depth\_min=None, depth\_max=None, rotation\_angle=0, shift\_north=0, shift\_east=0, log10=True, verbose=True)

Write out each depth slice as a raster in UTM coordinates.

Expecting

**a grid that is interpolated onto a regular grid of square cells with size cell\_size.** :param verbose:

Defaults to True.

**param log10**

Defaults to True.

**param shift\_east**

Defaults to 0.

**param shift\_north**

Defaults to 0.

---

**param cell\_size**  
Square cell size (cell\_size x cell\_size) in meters.

**type cell\_size**  
float

**param pad\_north**  
Number of padding cells to skip from outside in, if None defaults to self.pad\_north, defaults to None.

**type pad\_north**  
integer, optional

**param pad\_east**  
Number of padding cells to skip from outside in if None defaults to self.pad\_east, defaults to None.

**type pad\_east**  
integer, optional

**param save\_path**  
Path to save files to. If None use self.save\_path, defaults to None.

**type save\_path**  
string or Path, optional

**param depth\_min**  
Minimum depth to make raster for in meters,, defaults to None.

**type depth\_min**  
float, optional

**param depth\_max**  
Maximum depth to make raster for in meters,, defaults to None.

**type depth\_max**  
float, optional

**param rotation\_angle**  
Angle (degrees) to rotate the raster assuming clockwise positive rotation where North = 0, East = 90, defaults to 0.

**type rotation\_angle**  
float, optional

**raises ValueError**  
If utm\_epsg is not input.

**return**  
List of file paths to rasters.

**rtype**  
TYPE

**to\_ubc(basename)**  
Write a UBC .msh and .mod file. :param basename: :param save\_fn: DESCRIPTION. :type save\_fn: TYPE :return: DESCRIPTION. :rtype: TYPE

**to\_vtk(vtk\_fn=None, vtk\_save\_path=None, vtk\_fn\_basename='ModEM\_model\_res', \*\*kwargs)**  
Write a VTK file to plot in 3D rendering programs like Paraview. :param \*\*kwargs: :param vtk\_fn: Full path to VKT file to be written, defaults to None. :type vtk\_fn: string or Path, optional :param

vtk\_save\_path: Directory to save vtk file to, defaults to None. :type vtk\_save\_path: string or Path, optional :param vtk\_fn\_basename: Filename basename of vtk file, note that .vtr extension is automatically added, defaults to “ModEM\_model\_res”.

#### Parameters

- **geographic\_coordinates** (*boolean, optional*) – [ True | False ] True for geographic coordinates.
- **units** (*string, optional*) – Units of the spatial grid [ km | m | ft ], defaults to “km”.
- **coordinate\_system** – Coordinate system for the station, either the normal MT right-hand coordinate system with z+ down or the sinister z- down [ nez+ | enz- ], defaults to nez+.

#### Type

string

#### Returns

Full path to VTK file.

#### Return type

Path

### **to\_winglink\_out**(*save\_fn*)

Will write an .out file for LeapFrog.

Note that y is assumed to be S → N, e is assumed to be W → E and z is positive upwards. This means that index [0, 0, 0] is the southwest corner of the first layer. :param save\_fn: Full path to save file to. :type save\_fn: string or Path :return: DESCRIPTION. :rtype: TYPE

### **to\_ws3dinv\_intial**(*initial\_fn, res\_list=None*)

Write a WS3DINV intial model file.

### **to\_xarray**(\*\**kwargs*)

Put model in xarray format.

### **to\_xyres**(*save\_path=None, location\_type='EN', depth\_index='all', outfile\_basename='DepthSlice', log\_res=False, pad\_east=None, pad\_north=None*)

Write files containing depth slice data (x, y, res for each depth)

#### **origin = x,y coordinate of zero point of ModEM\_grid, or name of file**

containing this info (full path or relative to model files)

save\_path = path to save to, default is the model object save path location\_type = ‘EN’ or ‘LL’ xy points saved as eastings/northings or

longitude/latitude, if ‘LL’ need to also provide model\_epsg

model\_epsg = epsg number that was used to project the model outfile\_basename = string for basename for saving the depth slices. log\_res = True/False - option to save resistivity values as log10

instead of linear

clip = number of cells to clip on each of the east/west and north/south edges.

### **to\_xyzres**(*savefile=None, location\_type='EN', log\_res=False, pad\_east=None, pad\_north=None*)

Save a model file as a space delimited x y z res file.

**mtpy.processing package****Subpackages****mtpy.processing.aurora package****Submodules****mtpy.processing.aurora.process\_aurora module**

Created on Tue Jul 30 17:11:42 2024

@author: jpeacock

**class mtpy.processing.aurora.process\_aurora.AuroraProcessing(\*\*kwargs)**

Bases: *BaseProcessing*

Convenience class to process with Aurora

```
from mtpy.processing.aurora.process_aurora import AuroraProcessing

ap = AuroraProcessing()

set local station and path to MTH5
ap.local_station_id = "mt01"
ap.local_mth5_path = "/path/to/local_mth5.h5"

set remote station and path to MTH5
ap.remote_station_id = "rr01"
ap.remote_mth5_path = "/path/to/remote_mth5.h5"

process single sample rate
tf_obj = ap.process_single_sample_rate(sample_rate=1)

process multiple sample rates, merge them all together and
save transfer functions to the local MTH5
tf_processed_dict = ap.process(
 sample_rates=[4096, 1],
 merge=True,
 save_to_mth5=True
).
```

**create\_config(kernel\_dataset=None, decimation\_kwargs={}, \*\*kwargs)**

Decimation kwargs can include information about window,. :return: DESCRIPTION. :rtype: aurora.config

**create\_kernel\_dataset(run\_summary=None, local\_station\_id=None, remote\_station\_id=None, sample\_rate=None)**

This can be a stane alone method and return a kds, or create in place Build KernelDataset

**merge\_transfer\_functions(tf\_dict)**

Merge transfer functions according to AuroraProcessing.merge\_dict

**Parameters**

**tf\_dict** (*dict*) – dictionary of transfer functions

**Returns**

merged transfer function.

**Return type***mtpy.MT***process**(*sample\_rates=None*, *processing\_dict=None*, *merge=True*, *save\_to\_mth5=True*)

Need to either provide a list of sample rates to process or a processing dictionary.

If you provide just the sample rates, then at each sample rate a KernelDataset will be created as well as a subsequent config object which are then used to process the data.

If processing\_dict is set then the processing will loop through the dictionary and use the provided config and kernel datasets.

The processing dict has the following form

If merge is True then all runs for all sample rates are combined into a single function according to merge\_dict.

If save\_to\_mth5 is True then the transfer functions are saved to the local MTH5.

**Parameters**

- **sample\_rates** (*float or list, optional*) – list of sample rates to process, defaults to None
- **processing\_dict** (*dict, optional*) – processing dictionary as described above, defaults to None
- **merge** (*bool, optional*) – [ True | False ] True merges all sample rates into a single transfer function according to the merge\_dict, defaults to True
- **save\_to\_mth5** (*TYPE, optional*) – [ True | False ] save transfer functions to the local MTH5, defaults to True

**Raises**

- **ValueError** – If neither sample rates nor processing dict are provided
- **TypeError** – If the provided processing dictionary is not the correct format

**Returns**

dictionary of each sample rate processed in the form of {sample\_rate: {'processed': bool, 'tf': MT}}

**Return type**

dict

**process\_single\_sample\_rate**(*sample\_rate*, *config=None*, *kernel\_dataset=None*)

Process a single sample rate

**Parameters**

- **sample\_rate** (*float*) – sample rate of time series data
- **config** (*aurora.config, optional*) – configuration file, defaults to None
- **kernel\_dataset** (*mtpy.processing.KernelDataset, optional*) – Kernel dataset to define what data to process, defaults to None

**Returns**

transfer function

**Return type***mtpy.MT*

## Module contents

### Submodules

#### mtpy.processing.base module

Created on Tue Jul 30 17:13:07 2024

@author: jpeacock

**class mtpy.processing.base.BaseProcessing(\*\*kwargs)**

Bases: *KernelDataset*

Base processing class contains path to various files.

**get\_run\_summary()**

Get the RunSummary object. :param local: DESCRIPTION, defaults to True. :type local: TYPE, optional  
:param remote: DESCRIPTION, defaults to True. :type remote: TYPE, optional :return: DESCRIPTION.  
:rtype: TYPE

**has\_kernel\_dataset()**

Test if has kernel dataset. :return: DESCRIPTION. :rtype: TYPE

**has\_run\_summary()**

Check to see if run summary is set. :return: DESCRIPTION. :rtype: TYPE

**property mth5\_list**

Mth5 list. :return: List of mth5 to get run summary from. :rtype: list

**property run\_summary**

Run summary object. :return: DESCRIPTION. :rtype: TYPE

#### mtpy.processing.birrp module

### BIRRP

- deals with inputs and outputs from BIRRP

Created on Tue Sep 20 14:33:20 2016

@author: jrpeacock

**exception mtpy.processing.birrp.BIRRPPParameterError**

Bases: Exception

**class mtpy.processing.birrp.BIRRPPParameters(ilev=0, \*\*kwargs)**

Bases: object

Class to hold and produce the appropriate parameters given the input parameters.

**from\_dict(birrp\_dict)**

Set birrp parameters from dict.

**read\_config\_file(birrp\_config\_fn)**

Read in a configuration file and fill in the appropriate parameters.

**to\_dict()**

Get appropriate parameters.

```
write_config_file(save_fn)
```

Write a config file for birrp parameters.

```
class mtpy.processing.birrp.J2Edi(**kwargs)
```

Bases: object

Read in BIRRP out puts, in this case the .j file and convert that into an .edi file using the survey\_config\_fn parameters.

Key Word Arguments:

**birrp\_dir**

[string] full path to directory where birrp outputs are

**station**

[string] station name

**survey\_config\_fn**

[string] full path to survey configuration file with information on location and site setup must have a key that is the same as station.

**birrp\_config\_fn**

[string] full path to configuration file that was used to process with (all the birrp parameters used). If None is input, the file is searched for, if it is not found, the processing parameters are used from the .j file.

**j\_fn**

[string] full path to j file. If none is input the .j file is searched for in birrp\_dir.

Methods	Description
read_survey_config_fn	read in survey configuration file
get_birrp_config_fn	get the birrp_config_fn in birrp_dir
read_birrp_config_fn	read in birrp_config_fn
get_j_file	find .j file in birrp_dir
write_edi_file	write an .edi file fro all the provided information.

## Example

```
>>> import mtpy.proceessing.birrp as birrp
>>> j2edi_obj = birrp.J_To_Edi()
>>> j2edi_obj.birrp_dir = r"/home/data/mt01/BF/256"
>>> j2edi_obj.station = 'mt01'
>>> j2edi_obj.survey_config_fn = r"/home/data/2016_survey.cfg"
>>> j2edi_obj.write_edi_file()
```

**get\_birrp\_config\_fn()**

Get birrp configuration file from birrp directory.

**get\_j\_file(*birrp\_dir=None*)**

Get .j file output by birrp.

**read\_birrp\_config\_fn(*birrp\_config\_fn=None*)**

Read in birrp configuration file.

**read\_survey\_config\_fn(*survey\_config\_fn=None*)**

Read in survey configuration file and output into a useful dictionary.

```
write_edi_file(station=None, birrp_dir=None, survey_config_fn=None, birrp_config_fn=None,
copy_path=None)
```

Read in BIRRP out puts, in this case the .j file and convert that into an .edi file using the survey\_config\_fn parameters.

#### Parameters

- **\*\*station\*\*** – string name of station
- **\*\*birrp\_dir\*\*** – string full path to output directory for BIRRP
- **\*\*survey\_config\_fn\*\*** – string full path to survey configuration file
- **\*\*birrp\_config\_fn\*\*** – string full path to birrp configuration file *default* is none and is looked for in the birrp\_dir
- **\*\*copy\_path\*\*** – string full path to directory to copy the edi file to

Outputs:

**edi\_fn**  
[string] full path to edi file

The survey\_config\_fn is a file that has the structure:

```
[station]
b_instrument_amplification = 1 b_instrument_type = coil b_logger_gain = 1
b_logger_type = zen b_xaxis_azimuth = 0 b_yaxis_azimuth = 90 box = 26 date =
2015/06/09 e_instrument_amplification = 1 e_instrument_type = Ag-AgCl electrodes
e_logger_gain = 1 e_logger_type = zen e_xaxis_azimuth = 0 e_xaxis_length = 100
e_yaxis_azimuth = 90 e_yaxis_length = 100 elevation = 2113.2 hx = 2274 hy = 2284 hz
= 2254 lat = 37.7074236995 location = Earth lon = -118.999542099 network = USGS
notes = Generic config file rr_box = 25 rr_date = 2015/06/09 rr_hx = 2334 rr_hy = 2324
rr_lat = 37.6909139779 rr_lon = -119.028707542 rr_station = 302 sampling_interval
= all save_path = home\mtdatasurvey_01mt_01 station = 300 station_type = mt
```

This file can be written using mtpy.utils.configfile:

```
>>> import mtpy.utils.configfile as mtcfg
>>> station_dict = {}
>>> station_dict['lat'] = 21.346
>>> station_dict['lon'] = 122.45654
>>> station_dict['elev'] = 123.43
>>> cfg_fn = r"\home\mtdatasurvey_01"
>>> mtcfg.write_dict_to_configfile({station: station_dict}, cfg_fn)
```

```
class mtpy.processing.birrp.ScriptFile(script_fn=None, fn_arr=None, **kwargs)
```

Bases: *BIRRPParameters*

Class to read and write script file.

#### Arguments:

**fn\_arr**  
[numpy.ndarray] numpy.ndarray([[block 1], [block 2]])

#### Note

[block n] is a numpy structured array with data type

Name	Description	Type
fn	file path/name	string
nread	number of points to read	int
nskip	number of points to skip	int
comp	component	[ ex   ey   hx hy   hz ]
calibration_fn	calibration file path/name	string
rr	a remote reference channel	[ True   False ]
rr_num	remote reference pair number	int
start	start time iso format	Timestamp
stop	stop time iso format	Timestamp
station	station name	string
sampling_rate	sampling rate	int

**BIRRP Parameters:**

parameter	description
ilev	processing mode 0 for basic and 1 for advanced RR-2 stage
nout	Number of Output time series (2 or 3-> for BZ)
ninp	Number of input time series for E-field (1,2,3)
nref	Number of reference channels (2 for MT)
nrr	bounded remote reference (0) or 2 stage bounded influence (1)
tbw	Time bandwidth for Sepian sequence
deltat	Sampling rate (+) for (s), (-) for (Hz)
nfft	Length of FFT (should be even)
nsctinc	section increment divisor (2 to divide by half)
nsctmax	Number of windows used in FFT
nf1	1st frequency to extract from FFT window (>=3)
nfinc	frequency extraction increment
nfsect	number of frequencies to extract
mfft	AR filter factor, window divisor (2 for half)
uin	Quantile factor determination
ainlin	Residual rejection factor low end (usually 0)
ainuin	Residual rejection factor high end (.95-.99)
c2threshb	Coherence threshold for magnetics (0 if undesired)
c2threshe	Coherence threshold for electrics (0 if undesired)
nz	<b>Threshold for Bz (0=separate from E, 1=E threshold, 2=E and B)</b> Input if 3 B components else None
c2thresh1	Squared coherence for Bz, input if NZ=0, Nout=3
perlo	longest period to apply coherence threshold over

continues on next page

Table 12 – continued from previous page

parameter	description
perhi	shortes period to apply coherence threshold over
ofil	<b>Output file root(usually three letters, can add full path)</b>
nlev	Output files (0=Z; 1=Z,qq; 2=Z,qq,w; 3=Z,qq,w,d)
nprej	number of frequencies to reject
prej	frequencies to reject (+) for period, (-) for frequency
npcs	Number of independent data to be processed (1 for one segement)
nar	Prewhitening Filter (3< >15) or 0 if not desired',
imode	Output file mode (0=ascii; 1=binary; 2=headless ascii; 3=ascii in TS mode',
jmode	<b>input file mode (0=user defined; 1=sconvert2tart time YYYY-MM-DD HH:MM:SS)',</b>
nread	Number of points to be read for each data set (if segments>1 -> npts1,npts2...)',
nfil	<b>Filter parameters (0=none; &gt;0=input parameters; &lt;0=filename)</b>
nskip	<b>Skip number of points in time series (0) if no skip,</b> (if segements >1 -> input1,input2...)',
nskipr	Number of points to skip over (0) if none, (if segegments >1 -> input1,input2...)',
thetae	Rotation angles for electrics (relative to geo-magnetic North)(N,E,rot)',
thetab	Rotation angles for magnetics (relative to geo-magnetic North)(N,E,rot)',
thetar	Rotation angles for calculation (relative to geomagnetic North)(N,E,rot)'

 **Note**

Currently only supports jmode = 0 and imode = 0

 See also

BIRRP Manual and publications by Chave and Thomson for more details on the parameters found at:

<http://www.whoi.edu/science/AOPE/people/achave/Site/Next1.html>

**property comp\_list**

Comp list.

**property deltat**

Deltat function.

**make\_fn\_lines\_block\_00(fn\_arr)**

Make lines for file in script file which includes

- nread
- filter\_fn
- fn
- nskip.

**make\_fn\_lines\_block\_n(fn\_arr)**

Make lines for file in script file which includes

- nread
- filter\_fn
- fn
- nskip.

**property nout**

Nout function.

**property npcs**

Npcs function.

**property nref**

Nref function.

**write\_script\_file(script\_fn=None, ofil=None)**

Write script file.

**exception mtpy.processing.birrp.ScriptFileError**

Bases: Exception

**mtpy.processing.birrp.run(birrp\_exe, script\_file)**

Run a birrp script file from command line via python subprocess.

**Parameters**

- **\*\*birrp\_exe\*\*** – string full path to the compiled birrp executable
- **\*\*script\_file\*\*** – string full path to input script file following the guidelines of the BIRRP documentation.

Outputs:

**log\_file.log** : a log file of how BIRRP ran

#### See also

BIRRP Manual and publications by Chave and Thomson for more details on the parameters found at:

<http://www.whoi.edu/science/AOPE/people/achave/Site/Next1.html>

## mtpy.processing.filter module

mtpy/process/filter.py

Functions for the frequency filtering of raw time series.

@UofA, 2013 (LK)

Revised 2017 JP

`mtpy.processing.filter.adaptive_notch_filter(bx, df=100, notches=[50, 100], notchradius=0.5, freqrad=0.9, rp=0.1, dbstop_limit=5.0)`

Adaptive\_notch\_filter(bx, df, notches=[50,100], notchradius=.3, freqrad=.9) will apply a notch filter to the array bx by finding the nearest peak around the supplied notch locations. The filter is a zero-phase Chebyshev type 1 bandstop filter with minimal ripples.

#### Arguments::

**bx**

[np.ndarray(len\_time\_series)] time series to filter

**df**

[float] sampling frequency in Hz

**notches** : list of frequencies (Hz) to filter

**notchradius**

[float] radius of the notch in frequency domain (Hz)

**freqrad**

[float] radius to searching for peak about notch from notches

**rp**

[float] ripple of Chebyshev type 1 filter, lower numbers means less ripples

**dbstop\_limit**

[float (in decibels)] limits the difference between the peak at the notch and surrounding spectra. Any difference above dbstop\_limit will be filtered, anything less will not

#### Outputs:

```
bx : np.ndarray(len_time_series)
 filtered array

filtlst : list
 location of notches and power difference between peak of
 notch and average power.
```

. . Example: ::

(continues on next page)

(continued from previous page)

```
>>> import RemovePeriodicNoise_Kate as rmp
>>> # make a variable for the file to load in
>>> fn = r"/home/MT/mt01_20130101_000000.BX"
>>> # load in file, if the time series is not an ascii file
>>> # might need to add keywords to np.loadtxt or use another
>>> # method to read in the file
>>> bx = np.loadtxt(fn)
>>> # create a list of frequencies to filter out
>>> freq_notches = [50, 150, 200]
>>> # filter data
>>> bx_filt, filt_lst = rmp.adaptiveNotchFilter(bx, df=100.
>>> ... notches=freq_notches)
>>> #save the filtered data into a file
>>> np.savetxt(r"/home/MT/Filtered/mt01_20130101_000000.BX", bx_filt)
```

**Notes::**

Most of the time the default parameters work well, the only thing you need to change is the notches and perhaps the radius. I would test it out with a few time series to find the optimum parameters. Then make a loop over all your time series data. Something like

```
>>> import os
>>> dirpath = r"/home/MT"
>>> #make a director to save filtered time series
>>> save_path = r"/home/MT/Filtered"
>>> if not os.path.exists(save_path):
>>> os.mkdir(save_path)
>>> for fn in os.listdir(dirpath):
>>> bx = np.loadtxt(os.path.join(dirpath, fn))
>>> bx_filt, filt_lst = rmp.adaptiveNotchFilter(bx, df=100.
>>> ... notches=freq_notches)
>>> np.savetxt(os.path.join(save_path, fn), bx_filt)
```

`mtpy.processing.filter.butter_bandpass(lowcut, highcut, samplingrate, order=4)`

Butter bandpass.

`mtpy.processing.filter.butter_bandpass_filter(data, lowcut, highcut, samplingrate, order=4)`

Butter bandpass filter.

`mtpy.processing.filter.low_pass(f, low_pass_freq, cutoff_freq, sampling_rate)`

Low pass.

`mtpy.processing.filter.remove_periodic_noise(filename, dt, noiseperiods, save='n')`

RemovePeriodicNoise will take a window of length noise period and compute the median of signal for as many windows that can fit within the data. This median window is convolved with a series of delta functions at each window location to create a noise time series. This is then subtracted from the data to get a ‘noise free’ time series.

**Arguments::****filename**

[string (full path to file) or array] name of file to have periodic noise removed from can be an array

**dt**

[float] time sample rate (s)

**noiseperiods**

[list] a list of estimated periods with a range of values to look around [[noiseperiod1,df1],... ] where df1 is a fraction value find the peak about noiseperiod1 must be less than 1. (0 is a good start, but if you're periodic noise drifts, might need to adjust df1 to .2 or something)

**save**

[[ ‘y’ | ‘n’ ]]

- ‘y’ to save file to:

os.path.join(os.path.dirname(filename), ‘Filtered’, fn)

- ‘n’ to return the filtered time series

Outputs:

```
bxnf : np.ndarray
 filtered time series

pn : np.ndarray
 periodic noise time series

fitlst : list
 list of peaks found in time series
```

. Example::

```
>>> import RemovePeriodicNoise_Kate as rmp
>>> # make a variable for the file to load in
>>> fn = r"/home/MT/mt01_20130101_000000.BX"
>>> # filter data assuming a 12 second period in noise and save data
>>> rmp.remove_periodic_noise(fn, 100., [[12,0]], save='y')
```

**Notes::**

Test out the periodic noise period at first to see if it drifts. Then loop over files

```
>>> import os
>>> dirpath = r"/home/MT"
>>> for fn in os.listdir(dirpath):
>>> rmp.remove_periodic_noise(fn, 100., [[12,0]], save='y')
```

**mtpy.processing.filter.tukey(window\_length, alpha=0.2)**

The Tukey window, also known as the tapered cosine window, can be regarded as a cosine lobe of width alpha \* N / 2 that is convolved with a rectangle window of width (1 - alpha / 2). At alpha = 0 it becomes rectangular, and at alpha = 1 it becomes a Hann window.

output

Reference:

<http://www.mathworks.com/access/helpdesk/help/toolbox/signal/tukeywin.html>

**mtpy.processing.filter.zero\_pad(input\_array, power=2, pad\_fill=0)**

Pad the input array with pad\_fill to the next power of power.

For faster fft computation pad the array to the next power of 2 with zeros

**Arguments::**

**input\_array**  
[np.ndarray (only 1-d arrays are supported at the] moment)

**power**  
[[ 2 | 10 ]] power look for

**pad\_fill**  
[float or int] pad the array with this

**Output::**

**pad\_array** : np.ndarray padded with pad\_fill

## **mtpy.processing.kernel\_dataset module**

This module contains a class for representing a dataset that can be processed.

Development Notes:

Players on the stage: One or more mth5s.

Each mth5 has a “run\_summary” dataframe available. Run\_summary provides options for the local and possibly remote reference stations. Candidates for local station are the unique values in the station column.

For any candidate station, there are some integer n runs available. This yields  $2^n - 1$  possible combinations that can be processed, neglecting any flagging of time intervals within any run, or any joining of runs. (There are actually  $2^{n+1}$ , but we ignore the empty set, so -1)

Intuition suggests default ought to be to process n runs in n+1 configurations: {all runs} + each run individually. This will give a bulk answer, and bad runs can be flagged by comparing them. After an initial processing, the tfs can be reviewed and the problematic runs can be addressed.

The user can interact with the run\_summary\_df, selecting sub dataframes via querying, and in future maybe via some GUI (or a spreadsheet).

**The intended usage process is as follows:**

0. Start with a list of mth5s
1. Extract a run\_summary
2. Stare at the run\_summary\_df, and select a station “S” to process
3. Select a non-empty set of runs for station “S”
4. Select a remote reference “RR”, (this is allowed to be None)
5. Extract the sub-dataframe corresponding to acquisition\_runs from “S” and “RR” 7. If the remote is not None:
  - Drop the runs (rows) associated with RR that do not intersect with S
  - Restrict start/end times of RR runs that intersect with S so overlap is complete.
  - Restrict start/end times of S runs so that they intersect with remote
8. This is now a TFKernel Dataset Definition (ish). Initialize a default processing object and pass it this df. 

```
>>> cc = ConfigCreator()
>>> p = cc.create_from_kernel_dataset(kernel_dataset)
- Optionally pass emtf_band_file=emtf_band_setup_file
```

9. Edit the Processing Config appropriately,

TODO: Consider supporting a default value for ‘channel\_scale\_factors’ that is None,

TODO: Might need to groupby survey & station, for now consider station\_id unique.

---

```
class mtpy.processing.kernel_dataset.KernelDataset(df: DataFrame | None = None, local_station_id: str | None = "", remote_station_id: str | None = None, **kwargs)
```

Bases: object

This class is intended to work with mth5-derived channel\_summary or run\_summary dataframes, that specify time series intervals.

Development Notes: This class is closely related to (may actually be an extension of) RunSummary

The main idea is to specify one or two stations, and a list of acquisition “runs” that can be merged into a “processing run”. Each acquisition run can be further divided into non-overlapping chunks by specifying time-intervals associated with that acquisition run. An empty iterable of time-intervals associated with a run is interpreted as the interval corresponding to the entire run.

The time intervals can be used for several purposes but primarily: To specify contiguous chunks of data for: 1. STFT, that will be made into merged FC data structures 2. binding together into xarray time series, for eventual gap fill (and then STFT) 3. managing and analyse the availability of reference time series

The basic data structure can be represented as a table or as a tree: Station <- run <- [Intervals],

This is described in issue #118 <https://github.com/simpeg/aurora/issues/118>

Desired Properties a) This should be able to take a dictionary (tree) and return the tabular ( DataFrame) representation and vice versa. b) When there are two stations, can apply interval intersection rules, so that only time intervals when both stations are acquiring data are kept

From (a) above we can see that a simple table per station can represent the available data. That table can be generated by default from the mth5, and intervals to exclude some data can be added as needed.

(b) is really just the case of considering pairs of tables like (a)

Question: To return a copy or modify in-place when querying. Need to decide on standards and syntax. Handling this in general is messy because every function needs to be modified. Maybe better to use a decorator that allows for df kwarg to be passed, and if it is not passed the modification is done in place. The user who doesn't want to modify in place can work with a clone.

**add\_columns\_for\_processing()** → None

Add columns to the dataframe used during processing.

Development Notes: - This was originally in pipelines. - Q: Should mth5\_objs be keyed by survey-station?  
- A: Yes, and ... since the KernelDataset dataframe will be iterated over, should probably write an iterator method. This can iterate over survey-station tuples for multiple station processing. - Currently the model of keeping all these data objects “live” in the df seems to work OK, but is not well suited to HPC or lazy processing. :param mth5\_objs: Keys are station\_id, values are MTH5 objects. :type mth5\_objs: dict,

**clone()**

Return a deep copy.

**clone\_dataframe()** → DataFrame

Return a deep copy of dataframe.

**close\_mth5s()** → None

Loop over all unique mth5\_objs in dataset df and make sure they are closed.+.

**property df**

Df function.

**drop\_runs\_shorter\_than**(*minimum\_duration*: float, *units*: str | None = 's', *inplace*: bool | None = True)  
→ DataFrame

Drop runs from df that are inconsequentially short

Development Notes: This needs to have duration refreshed before hand. :param inplace:

Defaults to True.

#### Parameters

- **minimum\_duration** (*float*) – The minimum allowed duration for a run (in units of units).
- **units** (*Optional[str]*, *optional*) – Placeholder to support units that are not seconds, defaults to “s”.

**from\_run\_summary**(*run\_summary*: *RunSummary*, *local\_station\_id*: *str | None = None*, *remote\_station\_id*: *str | None = None*, *sample\_rate*: *float | int | None = None*) → *None*

Initialize the dataframe from a run summary. :param sample\_rate:

Defaults to None.

#### Parameters

- **run\_summary** (*RunSummary*) – Summary of available data for processing from one or more stations.
- **local\_station\_id** (*Optional[Union[str, None]]*, *optional*) – Label of the station for which an estimate will be computed, defaults to None.
- **remote\_station\_id** (*Optional[Union[str, None]]*, *optional*) – Label of the remote reference station, defaults to None.

**get\_run\_object**(*index\_or\_row*: *int | Series*) → *Run*

Gets the run object associated with a row of the df

Development Notes: TODO: This appears to be unused except by get\_station\_metadata.

Delete or integrate if desired. - This has likely been deprecated by direct calls to *run\_obj = row.mth5\_obj.from\_reference(row.run\_hdf5\_reference)* in pipelines..

#### Parameters

**index\_or\_row** (*Union[int, pd.Series]*)

#### Return run\_obj

The run associated with the row of the df.

#### Rtype run\_obj

*mt\_metadata.timeseries.Run*

**get\_station\_metadata**(*local\_station\_id*: *str*) → *Station*

Returns the station metadata.

Development Notes: TODO: This appears to be unused. Was probably a precursor to the

update\_survey\_metadata() method. Delete if unused. If used fill out doc:

“Helper function for archiving the TF – returns an object we can use to populate station metadata in the \_\_\_\_” :param local\_station\_id: The name of the local station. :type local\_station\_id: *str* :rtype: *mt\_metadata.timeseries.Station*

**has\_local\_mth5()** → bool

Test if local mth5 exists.

**has\_remote\_mth5()** → bool

Test if remote mth5 exists.

**initialize\_dataframe\_for\_processing()** → None

Adds extra columns needed for processing to the dataframe.

Populates them with mth5 objects, run\_hdf5\_reference, and xr.Datasets.

Development Notes: Note #1: When assigning xarrays to dataframe cells, df dislikes xr.Dataset, so we convert to xr.DataArray before packing df

**Note #2: [OPTIMIZATION] By accessing the run\_ts and packing the “run\_dataarray” column of the df, we**

perform a non-lazy operation, and essentially forcing the entire decimation\_level=0 dataset to be loaded into memory. Seeking a lazy method to handle this maybe worthwhile. For example, using a df.apply() approach to initialize only one row at a time would allow us to generate the FCs one row at a time and never ingest more than one run of data at a time ...

**Note #3: Uncommenting the continue statement here is desireable, will speed things up, but**

is not yet tested. A nice test would be to have two stations, some runs having FCs built and others not having FCs built. What goes wrong is in update\_survey\_metadata. Need a way to get the survey metadata from a run, not a run\_ts if possible

**initialize\_mth5s(mode: str | None = 'r')**

Returns a dict of open mth5 objects, keyed by station\_id

A future version of this for multiple station processing may need nested dict with [survey\_id][station].  
:return mth5\_objs: Keyed by stations.

local station id : mth5.mth5.MTH5 remote station id: mth5.mth5.MTH5.

**Rtype mth5\_objs**

dict

**property input\_channels: list**

Get input channels from data frame. :return: Input channels (sources). :rtype: list of strings

**property is\_single\_station: bool**

Returns True if no RR station.

**property local\_df: DataFrame**

Split data frame to just the local station runs. :return: Local station runs. :rtype: pd.DataFrame

**property local\_mth5\_path: Path**

Local mth5 path. :return: Local station MTH5 path, a property extracted from the dataframe. :rtype: Path

**property local\_station\_id: str**

Local station id.

**property local\_survey\_id: str**

Return string label for local survey id.

**property local\_survey\_metadata: Survey**

Return survey metadata for local station.

**property mini\_summary: DataFrame**

Return a dataframe that fits in terminal.

**property mth5\_objs**

Mth5 objs. :return: Dictionary [station\_id: mth5\_obj]. :rtype: dict

**property num\_sample\_rates: int**

Returns the number of unique sample rates in the dataframe.

**property output\_channels: list**

Get input channels from data frame. :return: Input channels (sources). :rtype: list of strings

**property processing\_id: str**

Its difficult to come up with unique ids without crazy long names so this is a generic id of local-remote, the station metadata will have run information and the config parameters.

**property remote\_df: DataFrame**

Split data frame to just the local station runs. :return: Local station runs. :rtype: pd.DataFrame

**property remote\_mth5\_path: Path**

Remote mth5 path. :return: Remote station MTH5 path, a property extracted from the dataframe. :rtype: Path

**property remote\_station\_id: str**

Remote station id.

**restrict\_run\_intervals\_to\_simultaneous(df: DataFrame) → None**

For each run in local\_station\_id check if it has overlap with other runs

There is room for optimization here

Note that you can wind up splitting runs here. For example, in that case where local is running continuously, but remote is intermittent. Then the local run may break into several chunks.. :rtype: None

**property sample\_rate: float**

Returns the sample rate that of the data in the dataframe.

**select\_station\_runs(station\_runs\_dict: dict, keep\_or\_drop: bool, inplace: bool | None = True) → DataFrame**

Partition the rows of df based on the contents of station\_runs\_dict and return one of the two partitions (based on value of keep\_or\_drop).

dict -> {station: [{run, start, end}]}{}

For example {"mt01": [{"0001", "0003"}]} :param inplace:

Defaults to True.

**Parameters**

- **station\_runs\_dict (dict)** – Keys are string ids of the stations to keep Values are lists of string labels for run\_ids to keep.
- **keep\_or\_drop (bool)** – If “keep”: returns df with only the station-runs specified in station\_runs\_dict If “drop”: returns df with station\_runs\_dict excised.
- **overwrite (bool)** – If True, self.df is overwritten with the reduced dataframe.

**Return type**

pd.DataFrame

---

**classmethod** `set_path`(*value: str | Path*) → *Path*

Set path.

**set\_run\_times**(*run\_time\_dict: dict, inplace: bool | None = True*)

Set run times from a dictionary formatted as {run\_id: {start, end}}. :param run\_time\_dict: DESCRIPTION. :type run\_time\_dict: dict :param inplace: DESCRIPTION, defaults to True. :type inplace: Optional[bool], optional :return: DESCRIPTION. :rtype: TYPE

**update\_survey\_metadata**(*i: int, row: Series, run\_ts: RunTS*) → *None*

Wrangle survey\_metadata into kernel\_dataset.

Development Notes: - The survey metadata needs to be passed to TF before exporting data. - This was factored out of initialize\_dataframe\_for\_processing - TODO: It looks like we don't need to pass the whole run\_ts, just its metadata

There may be some performance implications to passing the whole object. Consider passing run\_ts.survey\_metadata, run\_ts.run\_metadata, run\_ts.station\_metadata only

### Parameters

- **i (int)** – This would be the index of row, if we were sure that the dataframe was cleanly indexed.
- **row (pd.Series)**
- **run\_ts (mth5.timeseries.run\_ts.RunTS)** – Mth5 object having the survey\_metadata.

### Return type

*None*

`mtpy.processing.kernel_dataset.intervals_overlap`(*start1: Timestamp, end1: Timestamp, start2: Timestamp, end2: Timestamp*) → *bool*

Checks if intervals 1, and 2 overlap.

Interval 1 is (start1, end1), Interval 2 is (start2, end2),

Development Notes: This may work vectorized out of the box but has not been tested. Also, it is intended to work with pd.Timestamp objects, but should work for many objects that have an ordering associated. This website was used as a reference when writing the method: <https://stackoverflow.com/questions/3721249/python-date-interval-intersection> :param start1: Start of interval 1. :type start1: pd.Timestamp :param end1: End of interval 1. :type end1: pd.Timestamp :param start2: Start of interval 2. :type start2: pd.Timestamp :param end2: End of interval 2. :type end2: pd.Timestamp :return cond: True if the intervals overlap, False if they do now. :rtype cond: bool

`mtpy.processing.kernel_dataset.overlap`(*t1\_start: Timestamp, t1\_end: Timestamp, t2\_start: Timestamp, t2\_end: Timestamp*) → *tuple*

Get the start and end times of the overlap between two intervals.

Interval 1 is (start1, end1), Interval 2 is (start2, end2),

### Development Notes:

Possibly some nicer syntax in this discussion: <https://stackoverflow.com/questions/3721249/python-date-interval-intersection> - Intended to work with pd.Timestamp objects, but should work for many objects

that have an ordering associated.

### Parameters

- **t1\_start** (*pd.Timestamp*) – The start of interval 1.
- **t1\_end** (*pd.Timestamp*) – The end of interval 1.
- **t2\_start** (*pd.Timestamp*) – The start of interval 2.
- **t2\_end** (*pd.Timestamp*) – The end of interval 2.

**Return start, end**

Start, end are either same type as input, or they are None, None.

**Rtype start, end**

tuple

```
mtpy.processing.kernel_dataset.restrict_to_station_list(df: DataFrame, station_ids: str | list,
inplace: bool | None = True) → DataFrame
```

Drops all rows of run\_summary dataframe where station\_ids are NOT in the provided list of station\_ids. Operates on a deepcopy of self.df if a df isn't provided :param df: A run summary dataframer. :type df: pd.DataFrame :param station\_ids: These are the station ids to keep, normally local and remote. :type station\_ids: Union[str, list] :param inplace: If True, self.df is overwritten with the reduced dataframe, defaults to True. :type inplace: Optional[bool], optional :rtype: pd.DataFrame

## **mtpy.processing.run\_summary module**

This module contains the RunSummary class.

This is a helper class that summarizes the Runs in an mth5.

TODO: This class and methods could be replaced by methods in MTH5.

Functionality of RunSummary() 1. User can get a list of local\_station options, which correspond to unique pairs of values: (survey, station)

2. User can see all possible ways of processing the data: - one list per (survey, station) pair in the run\_summary

Some of the following functionalities may end up in KernelDataset: 3. User can select local\_station -this can trigger a reduction of runs to only those that are from the local staion and simultaneous runs at other stations 4. Given a local station, a list of possible reference stations can be generated 5. Given a remote reference station, a list of all relevant runs, truncated to maximize coverage of the local station runs is generated 6. Given such a “restricted run list”, runs can be dropped 7. Time interval endpoints can be changed

**Development Notes:****TODO: consider adding methods:**

- drop\_runs\_shorter\_than”: removes short runs from summary
- fill\_gaps\_by\_time\_interval”: allows runs to be merged if gaps between are short
- fill\_gaps\_by\_run\_names”: allows runs to be merged if gaps between are short

TODO: Consider whether this should return a copy or modify in-place when querying the df.

```
class mtpy.processing.run_summary.RunSummary(input_dict: dict | None = None, df: DataFrame | None =
None)
```

Bases: object

Class to contain a run-summary table from one or more mth5s.

WIP: For the full MMT case this may need modification to a channel based summary.

**clone()**

2022-10-20: Cloning may be causing issues with extra instances of open h5 files ...

**property df: DataFrame**

Df function.

**drop\_no\_data\_rows() → bool**

Drops rows marked *has\_data* = False and resets the index of self.df.

**from\_mth5s(mth5\_list) → list**

Iterates over mth5s in list and creates one big dataframe summarizing the runs

**property mini\_summary: DataFrame**

Shows the dataframe with only a few columns for readability.

**property print\_mini\_summary: str**

Calls minisummary through logger so it is formatted.

**set\_sample\_rate(sample\_rate: float, inplace: bool = False)**

Set the sample rate so that the run summary represents all runs for a single sample rate.

**Parameters**

- **sample\_rate** (*float*)
- **inplace** (*bool, optional*) – DESCRIPTION. By default, False.

**Returns**

DESCRIPTION.

**Return type**

TYPE

```
mtpy.processing.run_summary.extract_run_summaries_from_mth5s(mth5_list, summary_type='run',
 deduplicate=True)
```

Given a list of mth5's, iterate over them, extracting run\_summaries and merging into one big table.

Development Notes: ToDo: Move this method into mth5? or mth5\_helpers? ToDo: Make this a class so that the `__repr__` is a nice visual representation of the df, like what channel summary does in mth5 - 2022-05-28 Modified to allow this method to accept mth5 objects as well as the already supported types of pathlib.Path or str

In order to drop duplicates I used the solution here: <https://stackoverflow.com/questions/43855462/pandas-drop-duplicates-method-not-working-on-dataframe-containing-lists>

**Parameters**

- **deduplicate** (, *defaults to True.* : *bool, optional*) – By default, True.
- **mth5\_list**
- **mth5\_paths** (*list*) – Paths or strings that point to mth5s.
- **summary\_type** (*string, optional*) – One of [“channel”, “run”] “channel” returns concatenated channel summary, “run” returns concatenated run summary,. By default, “run”.
- **deduplicate**

**Returns**

**super\_summary** – Given a list of mth5s, a dataframe of all available runs.

**Return type**

pd.DataFrame

```
mtpy.processing.run_summary.extract_run_summary_from_mth5(mth5_obj, summary_type: str | None = 'run')
```

Given a single mth5 object, get the channel\_summary and compress it to a run\_summary.

Development Notes: TODO: Move this into MTH5 or replace with MTH5 built-in run\_summary method.

**Parameters**

- **mth5\_obj** (*mth5.mth5.MTH5*) – The initialized mth5 object that will be interrogated.
- **summary\_type** (*Optional[str]*, *optional*) – One of [“run”, “channel”]. Returns a run summary or a channel summary. By default, “run”.

**Returns**

**out\_df** – Table summarizing the available runs in the input mth5\_obj.

**Return type**

pd.DataFrame

## mtpy.processing.tf module

mtpy/processing/tf.py

Functions for the time-frequency analysis of time series data.

Output can be visualised with the help of mtpy/imaging/spectrogram.py

JP, 2013

```
mtpy.processing.tf.dctrend(f)
```

dctrend(f) will remove a dc trend from the function f.

**Arguments:**

**f**

[np.ndarray()] array to remove dc trend from

**Returns:**

**fdc**

[np.ndarray()] array f with dc component removed

```
mtpy.processing.tf.decimate(f, m, window_function='hanning')
```

resamples the data at the interval m so that the returned array is len(f)/m samples long

**Arguments:**

**f**

[np.ndarray] array to be decimated

**m**

[int] decimation factor

**window\_function**

[windowing function to apply to the data] to make sure there is no Gibbs ringing or aliasing see  
scipy.signal.window for all the options

**Returns:****fdec**

[np.ndarray()] array f decimated by factor m

**mtpy.processing.tf.dwindow(window)**

Calculates the derivative of the given window. Used for reassignment methods

**Arguments:****window**

[np.ndarray] some sort of windowed array

**Returns:****dwin**

[np.ndarray] derivative of window

**mtpy.processing.tf.gausswin(window\_len, alpha=2.5)**

gausswin will compute a gaussian window of length winlen with a variance of alpha

**Arguments:****window\_len: int**

length of desired window

**alpha**

[float] 1/standard deviation of window, ie full width half max of window

**Returns:****gauss\_window**

[np.array] gaussian window

**mtpy.processing.tf.modifiedb(fx, tstep=32, nfbins=1024, df=1.0, nh=255, beta=0.2)**

Calculates the modified b distribution as defined by  $\cosh(n)^{-2}$  beta for an array fx. Supposed to remove cross terms in the WVD.

**Arguments:****fx**

[list or np.ndarray] the function to have a spectrogram computed for cross-correlation input as [fx1, fx2]

**nh**

[int (should be odd)] window length for each time step *default* is None and window is calculated automatically

**tstep**

[int] number of sample between short windows *default* is  $2^{**5} = 32$

**df**

[float] sampling frequency

**nfbins**

[int (should be power of 2 and equal or larger than nh)] number of frequency bins

**beta**

[float] smoothing coefficient ussully between [0, 1]

**Returns:**

**tfarray**

[np.ndarray(nfbins/2, len(fx)/tstep)] SPWVD spectrogram in units of amplitude

**tlist**

[np.array()] array of time instances for each window calculated

**flist**

[np.ndarray(nfbins/2)] frequency array containing only positive frequencies where the Fourier coefficients were calculated

`mtpy.processing.tf.normalize_L2(f)`

`normalize_L2(f)` returns the array f normalized by the L2 norm ->  $f / (\sqrt{\sum(\text{abs}(x_i)^2)})$ .

**Arguments**

**f**

[np.ndarray()] array to be normalized

**Returns:**

**fnorm**

[np.ndarray()] array f normalized in L2 sense

`mtpy.processing.tf.padzeros(f, npad=None, pad_pattern=None)`

`padzeros(f)` returns a function that is padded with zeros to the next power of 2 for faster processing for fft or to length npad if given.

**Arguments:**

**f**

[np.ndarray(m, n)] array to pad

**npad**

[int] length to pad to if None finds next power of 2

**pad\_pattern**

[int or float] pattern to pad with if None set zeros

**Returns:**

**fpad**

[np.ndarray(m, npad)] array f padded to length npad with pad\_pattern

**Example**

```
:: >>> x_array = np.sin(np.arange(0, 2, .01)*np.pi/3) >>> print len(x_array) >>> x_array_pad = padzeros(x_array) >>> print len(x_array_pad)
```

`mtpy.processing.tf.reassigned_smethod(fx, nh=127, tstep=16, nfbins=512, df=1.0, alpha=4, thresh=0.01, L=5)`

Calculates the reassigned S-method as described by Djurovic[1999] by using the spectrogram to estimate the reassignment.

**Arguments:**

for cross-correlation input as [fx1, fx2]

**L**

[int (should be odd)] length of window for S-method calculation, higher numbers tend toward WVD

**nh**

[int (should be power of 2)] window length for each time step *default* is  $2^{**8} = 256$

**alpha**

[float] inverse of full-width half max of gaussian window, smaller numbers mean broader windows.

**thresh**

[float] threshold for reassignment, lower numbers more points reassigned, higher numbers less points reassigned

**tstep**

[int] number of sample between short windows *default* is  $2^{**7} = 128$

**df**

[float] sampling frequency

**nfbins**

[int (should be power of 2 and equal or larger than nh)] number of frequency bins

**Returns:****tfarray**

[np.ndarray(nfbins/2, len(fx)/tstep)] S-method spectrogram in units of amplitude

**tlst**

[np.array()] array of time instances for each window calculated

**flist**

[np.ndarray(nfbins/2)] frequency array containing only positive frequencies where the Fourier coefficients were calculated

**sm**

[np.ndarray(nfbins/2, len(fx)/tstep)] S-method spectrogram in units of amplitude

`mtpy.processing.tf.reassigned_stft(fx, nh=63, tstep=32, nfbins=1024, df=1.0, alpha=4, threshold=None)`

Computes the reassigned spectrogram by estimating the center of gravity of the signal and condensing dispersed energy back to that location. Works well for data with minimal noise and strong spectral structure.

**Arguments:****fx**

[np.ndarray] time series to be analyzed

**nh**

[int(should be odd)] length of gaussian window that is applied to the short time intervals *default* is 127

**tstep**

[int] time step for each window calculation *default* is 64

**nfbins**

[int (should be a power of 2 and larger or equal to nh) number of frequency bins to calculate, note result will be length nfbins/2 *default* is 1024]

**df**

[float or int] sampling frequency (Hz)

**alpha**

[float] reciprocal of full width half max of gaussian window *default* is 4

**threshold**

[float] threshold value for reassignment If None the threshold is automatically calculated *default* is None

**returns**

**np.ndarray(nfbins/2, len(fx)/tstep)**

reassigned spectrogram in units of amplitude

**tlst**

[np.array()] array of time instances for each window calculated

**flst**

[np.ndarray(nfbins/2)] frequency array containing only positive frequencies where the Fourier coefficients were calculated

**stft**

[np.ndarray(nfbins/2, len(fx)/tstep)] standard spectrogram calculated from stft in units of amplitude

**rtype**

**rtfarray**

```
mtpy.processing.tf.robust_smethod(fx, L=5, nh=128, tstep=32, nfbins=1024, df=1.0, robusttype='median',
sigmal=None, alpha=0.325)
```

Computes the robust Smethod via the robust spectrogram.

**Arguments:**

**fx**

[list or np.ndarray] the function to have a spectrogram computed for cross-correlation input as [fx1, fx2]

**L**

[int (should be odd)] length of window for S-method calculation, higher numbers tend toward WVD

**nh**

[int (should be power of 2)] window length for each time step *default* is  $2^{**}8 = 256$

**ng**

[int (should be odd)] length of smoothing window along frequency plane

**tstep**

[int] number of sample between short windows *default* is  $2^{**}7 = 128$

**df**

[float] sampling frequency

**nfbins**

[int (should be power of 2 and equal or larger than nh)] number of frequency bins

**robusttype**

[[ ‘median’ | ‘L’ ]] type of robust STFT to compute. *default* is ‘median’

**simgaL**

[float] full-width half max of gaussian window applied in frequency

**Returns:****tfarray**

[np.ndarray(nfbins/2, len(fx)/tstep)] S-method spectrogram in units of amplitude

**tlst**

[np.array()] array of time instances for each window calculated

**f1st**

[np.ndarray(nfbins/2)] frequency array containing only positive frequencies where the Fourier coefficients were calculated

**pxx**

[np.ndarray(nfbins/2, len(fx)/tstep)] STFT spectrogram in units of amplitude

`mtpy.processing.tf.robust_stft_L(fx, alpha=0.325, nh=256, tstep=32, df=1.0, nfbins=1024)`

Calculates the robust spectrogram by estimating the vector median and summing terms estimated by alpha coefficients.

**Arguments:****fx**

[list or np.ndarray] the function to have a spectrogram computed for cross-correlation input as [fx1, fx2]

**alpha**

[float] robust parameter [0,.5] -> 0 gives spectrogram, 0.5 gives median stft *default* is 0.325

**nh**

[int (should be power of 2)] window length for each time step *default* is  $2^{**}8 = 256$

**tstep**

[int] number of sample between short windows *default* is  $2^{**}7 = 128$

**df**

[float] sampling frequency

**nfbins**

[int (should be power of 2 and equal or larger than nh)] number of frequency bins

**Returns:****tfarray**

[np.ndarray(nfbins/2, len(fx)/tstep)] spectrogram in units of amplitude

**tlst**

[np.array()] array of time instances for each window calculated

**f1st**

[np.ndarray(nfbins/2)] frequency array containing only positive frequencies where the Fourier coefficients were calculated

```
mtpy.processing.tf.robust_stft_median(fx, nh=256, tstep=32, df=1.0, nfbins=1024)
```

Calculates the robust spectrogram using the vector median simplification.

#### Arguments:

**fx**

[list or np.ndarray] the function to have a spectrogram computed for cross-correlation input as [fx1, fx2]

**nh**

[int (should be power of 2)] window length for each time step *default* is  $2^{**8} = 256$

**tstep**

[int] number of sample between short windows *default* is  $2^{**7} = 128$

**df**

[float] sampling frequency (Hz)

**nfbins**

[int (should be power of 2 and equal or larger than nh)] number of frequency bins

#### Returns:

**tfarray**

[np.ndarray(nfbins/2, len(fx)/tstep)] spectrogram in units of amplitude

**tlst**

[np.array()] array of time instances for each window calculated

**flst**

[np.ndarray(nfbins/2)] frequency array containing only positive frequencies where the Fourier coefficients were calculated

```
mtpy.processing.tf.robust_wvd(fx, nh=127, ng=15, tstep=16, nfbins=256, df=1.0, sigmat=None,
sigmag=None)
```

Calculate the robust Wigner-Ville distribution for an array fx. Smoothed with Gaussians windows to get best localization.

#### Arguments:

**fx**

[list or np.ndarray] the function to have a spectrogram computed for cross-correlation input as [fx1, fx2]

**nh**

[int (should be power of 2)] window length for each time step *default* is  $2^{**8} = 256$

**tstep**

[int] number of sample between short windows *default* is  $2^{**7} = 128$

**ng**

[int (should be odd)] length of smoothing window along frequency plane *default* is  $2^{**4}-1 = 15$

**df**

[float] sampling frequency

**nfbins**

[int (should be power of 2 and equal or larger than nh)] number of frequency bins

**sigmat**

[float] std of window h, ie full width half max of gaussian *default* is None and sigmat is calculate automatically

**sigmaf**

[float] std of window g, ie full width half max of gaussian *default* is None and sigmaf is calculate automatically

**Returns:****tfarray**

[np.ndarray(nfbins/2, len(fx)/tstep)] spectrogram in units of amplitude

**tlst**

[np.array()] array of time instances for each window calculated

**flst**

[np.ndarray(nfbins/2)] frequency array containing only positive frequencies where the Fourier coefficients were calculated

`mtpy.processing.tf.sinc_filter(f, fcutoff=10.0, w=10.0, dt=0.001)`

Applies a sinc filter of width w to the function f by multiplying in the frequency domain.

**Arguments:****f**

[np.ndarray()] array to filter

**fcutoff**

[float] cutoff frequency of sinc filter

**w**

[float] length of filter

**dt**

[float] sampling rate in time (s)

**Returns:****f\_filt**

[np.ndarray()] f with sinc filter applied

`mtpy.processing.tf.smethod(fx, L=11, nh=256, tstep=128, ng=1, df=1.0, nfbins=1024, sigmaL=None)`

Calculates the smethod by estimating the STFT first and computing the WV of window length L in the frequency domain.

For larger L more of WV estimation, if L=0 get back STFT

**Arguments:****fx**

[list or np.ndarray] the function to have a spectrogram computed for cross-correlation input as [fx1, fx2]

**L**

[int (should be odd)] length of window for S-method calculation, higher numbers tend toward WVD

**nh**

[int (should be power of 2)] window length for each time step *default* is  $2^{**}8 = 256$

**ng**

[int (should be odd)] length of smoothing window along frequency plane

**tstep**

[int] number of sample between short windows *default* is  $2^{**}7 = 128$

**df**

[float] sampling frequency

**nfbins**

[int (should be power of 2 and equal or larger than nh)] number of frequency bins

**Returns:****tfarray**

[np.ndarray(nfbins/2, len(fx)/tstep)] S-method spectrogram in units of amplitude

**tlst**

[np.array()] array of time instances for each window calculated

**flst**

[np.ndarray(nfbins/2)] frequency array containing only positive frequencies where the Fourier coefficients were calculated

**pxx**

[np.ndarray(nfbins/2, len(fx)/tstep)] STFT spectrogram in units of amplitude

`mtpy.processing.tf.specwv(fx, tstep=32, nfbins=1024, nhs=256, nhwv=511, ngwv=7, df=1.0)`

Calculates the Wigner-Ville distribution multiplied by the STFT windowed by the common gaussian window h for an array f. Handy for removing cross terms in the wvd.

**Arguments:****fx**

[list or np.ndarray] the function to have a spectrogram computed for

**tstep**

[int]

number of sample between short windows *default* is  $2^{**}7 = 128$

**nhs**

[int (should be power of 2)] window length for each time step to calculate STFT *default* is  $2^{**}8 = 256$  and window is calculated automatically

**nhwv**

[int (should be odd)] length of smoothing window for each time step to calculate WVD. *default* is  $2^{**}9-1 = 511$

**ngwv**

[int (should be odd)] length of frequency smoothing window for each time step to calculate WVD. *default* is  $2^{**}3-1 = 7$

**df**

[float] sampling frequency

**nfbins**

[int (should be power of 2 and equal or larger than nh)] number of frequency bins

**Returns:****tfarray**

[np.ndarray(nfbins/2, len(fx)/tstep)] SPWVD spectrogram in units of amplitude

**tlist**

[np.array()] array of time instances for each window calculated

**flist**

[np.ndarray(nfbins/2)] frequency array containing only positive frequencies where the Fourier coefficients were calculated

```
mtpy.processing.tf.spwvd(fx, tstep=32, nfbins=1024, df=1.0, nh=None, ng=None, sigmat=None,
 sigmaf=None)
```

Calculates the smoothed pseudo Wigner-Ville distribution for an array fx. Smoothed with Gaussians windows to get best localization.

Can be input as [fx1, fx2] to compute cross spectra.

**Arguments:****fx**

[list or np.ndarray] the function to have a spectrogram computed for for cross-correlation input as [fx1, fx2]

**nh**

[int (should be odd)] window length for each time step *default* is None and window is calculated automatically

**tstep**

[int] number of sample between short windows *default* is  $2^{**}7 = 128$

**ng**

[int (should be odd)] length of smoothing window along frequency plane

**df**

[float] sampling frequency

**nfbins**

[int (should be power of 2 and equal or larger than nh)] number of frequency bins

**sigmat**

[float] std of window h, ie full width half max of gaussian *default* is None and sigmat is calculate automatically

**sigmaf**

[float] std of window g, ie full width half max of gaussian *default* is None and sigmaf is calculate automatically

**Returns:****tfarray**

[np.ndarray(nfbins/2, len(fx)/tstep)] SPWVD spectrogram in units of amplitude

**tlist**

[np.array()] array of time instances for each window calculated

**fist**

[np.ndarray(nfbins/2)] frequency array containing only positive frequencies where the Fourier coefficients were calculated

```
mtpy.processing.tf.stfbss(X, nsources=5, ng=31, nh=511, tstep=63, df=1.0, nfbins=1024, tftol=1e-08, L=7,
normalize=True, tftype='spwvd', alpha=0.38)
```

```
btfssX,nsources=5,ng=2**5-1,nh=2**9-1,tstep=2**6-1,df=1.0,nfbins=2**10,
tftol=1.E-8,normalize=True)
```

estimates sources using a blind source algorithm based on spatial time-frequency distributions. At the moment this algorithm uses the SPWVD to estimate TF distributions.

**Arguments**

**X = m x n array of time series, where m is number of time series and n**  
is length of each time series

nsources = number of estimated sources ng = frequency window length nh = time window length tstep = time step increment df = sampling frequency (Hz) nfbins = number of frequencies tftol = tolerance for a time-frequency point to be estimated as a cross

term or as an auto term, the higher the number the more auto terms.

**normalization = True or False, True to normalize, False if already**  
normalized

**Returns**

Se = estimated individual signals up to a permutation and scale Ae = estimated mixing matrix as X=A\*S

```
mtpy.processing.tf.stft(fx, nh=256, tstep=128, ng=1, df=1.0, nfbins=1024)
```

calculate the spectrogram of the given function by calculating the fft of a window of length nh at each time instance with an interval of tstep. The frequency resolution is nfbins.

Can compute the cross STFT by inputting fx as [fx1, fx2]

**Arguments:****fx**

[list or np.ndarray] the function to have a spectrogram computed for for cross-correlation input as [fx1, fx2]

**nh**

[int (should be power of 2)] window length for each time step *default* is 2\*\*8 = 256

**tstep**

[int] number of sample between short windows *default* is 2\*\*7 = 128

**ng**

[int (should be odd)] length of smoothing window along frequency plane

**df**

[float] sampling frequency

**nfbins**

[int (should be power of 2 and equal or larger than nh)] number of frequency bins

**Returns:****tfarray**

[np.ndarray(nfbins/2, len(fx)/tstep)] spectrogram in units of amplitude

**tlst**

[np.array()] array of time instances for each window calculated

**flist**

[np.ndarray(nfbins/2)] frequency array containing only positive frequencies where the Fourier coefficients were calculated

`mtpy.processing.tf.wvd(fx, nh=255, tstep=32, nfbins=1024, df=1.0)`

calculates the Wigner-Ville distribution of f.

Can compute the cross spectra by inputting fx as [fx1,fx2]

**Arguments:****fx**

[list or np.ndarray] the function to have a spectrogram computed for for cross-correlation input as [fx1, fx2]

**nh**

[int (should be odd)] window length for each time step *default* is  $2^{**8}-1 = 255$

**tstep**

[int] number of sample between short windows *default* is  $2^{**7} = 128$

**df**

[float] sampling frequency

**nfbins**

[int (should be power of 2 and equal or larger than nh)] number of frequency bins

**Returns:****tfarray**

[np.ndarray(nfbins/2, len(fx)/tstep)] spectrogram in units of amplitude

**tlst**

[np.array()] array of time instances for each window calculated

**flist**

[np.ndarray(nfbins/2)] frequency array containing only positive frequencies where the Fourier coefficients were calculated

`mtpy.processing.tf.wvd_analytic_signal(fx)`

Computes the analytic signal for WVVD as defined by J. M. O' Toole, M. Mesbah, and B. Boashash, (2008), "A New Discrete Analytic Signal for Reducing Aliasing in the

Discrete Wigner-Ville Distribution", IEEE Trans. on Signal Processing,

**Argument:****fx**

[np.ndarray()] signal to compute analytic signal for with length N

**Returns:****fxa**

[np.ndarray()] analytic signal of fx with length 2\*N

**Module contents****class mtpy.processing.AuroraProcessing(\*\*kwargs)**Bases: *BaseProcessing*

Convenience class to process with Aurora

```
from mtpy.processing.aurora.process_aurora import AuroraProcessing

ap = AuroraProcessing()

set local station and path to MTH5
ap.local_station_id = "mt01"
ap.local_mth5_path = "/path/to/local_mth5.h5"

set remote station and path to MTH5
ap.remote_station_id = "rr01"
ap.remote_mth5_path = "/path/to/remote_mth5.h5"

process single sample rate
tf_obj = ap.process_single_sample_rate(sample_rate=1)

process multiple sample rates, merge them all together and
save transfer functions to the local MTH5
tf_processed_dict = ap.process(
 sample_rates=[4096, 1],
 merge=True,
 save_to_mth5=True
).
```

**create\_config(kernel\_dataset=None, decimation\_kwargs={}, \*\*kwargs)**

Decimation kwargs can include information about window,. :return: DESCRIPTION. :rtype: aurora.config

**create\_kernel\_dataset(run\_summary=None, local\_station\_id=None, remote\_station\_id=None, sample\_rate=None)**

This can be a stane alone method and return a kds, or create in place Build KernelDataset

**merge\_transfer\_functions(tf\_dict)**

Merge transfer functions according to AuroraProcessing.merge\_dict

**Parameters****tf\_dict** (*dict*) – dictionary of transfer functions**Returns**

merged transfer function.

**Return type***mtpy.MT*

**process**(*sample\_rates=None*, *processing\_dict=None*, *merge=True*, *save\_to\_mth5=True*)

Need to either provide a list of sample rates to process or a processing dictionary.

If you provide just the sample rates, then at each sample rate a KernelDataset will be created as well as a subsequent config object which are then used to process the data.

If processing\_dict is set then the processing will loop through the dictionary and use the provided config and kernel datasets.

The processing dict has the following form

If merge is True then all runs for all sample rates are combined into a single function according to merge\_dict.

If save\_to\_mth5 is True then the transfer functions are saved to the local MTH5.

**Parameters**

- **sample\_rates** (*float or list, optional*) – list of sample rates to process, defaults to None
- **processing\_dict** (*dict, optional*) – processing dictionary as described above, defaults to None
- **merge** (*bool, optional*) – [ True | False ] True merges all sample rates into a single transfer function according to the merge\_dict, defaults to True
- **save\_to\_mth5** (*TYPE, optional*) – [ True | False ] save transfer functions to the local MTH5, defaults to True

**Raises**

- **ValueError** – If neither sample rates nor processing dict are provided
- **TypeError** – If the provided processing dictionary is not the correct format

**Returns**

dictionary of each sample rate processed in the form of {sample\_rate: {‘processed’: bool, ‘tf’: MT}}

**Return type**

dict

**process\_single\_sample\_rate**(*sample\_rate*, *config=None*, *kernel\_dataset=None*)

Process a single sample rate

**Parameters**

- **sample\_rate** (*float*) – sample rate of time series data
- **config** (*aurora.config, optional*) – configuration file, defaults to None
- **kernel\_dataset** (*mtpy.processingKernelDataset, optional*) – Kernel dataset to define what data to process, defaults to None

**Returns**

transfer function

**Return type**

*mtpy.MT*

**class mtpy.processingKernelDataset**(*df: DataFrame | None = None*, *local\_station\_id: str | None = ”*, *remote\_station\_id: str | None = None*, *\*\*kwargs*)

Bases: `object`

This class is intended to work with mth5-derived channel\_summary or run\_summary dataframes, that specify time series intervals.

Development Notes: This class is closely related to (may actually be an extension of) RunSummary

The main idea is to specify one or two stations, and a list of acquisition “runs” that can be merged into a “processing run”. Each acquisition run can be further divided into non-overlapping chunks by specifying time-intervals associated with that acquisition run. An empty iterable of time-intervals associated with a run is interpreted as the interval corresponding to the entire run.

The time intervals can be used for several purposes but primarily: To specify contiguous chunks of data for: 1. STFT, that will be made into merged FC data structures 2. binding together into xarray time series, for eventual gap fill (and then STFT) 3. managing and analyse the availability of reference time series

The basic data structure can be represented as a table or as a tree: Station <- run <- [Intervals],

This is described in issue #118 <https://github.com/simpeg/aurora/issues/118>

Desired Properties a) This should be able to take a dictionary (tree) and return the tabular ( DataFrame) representation and vice versa. b) When there are two stations, can apply interval intersection rules, so that only time intervals when both stations are acquiring data are kept

From (a) above we can see that a simple table per station can represent the available data. That table can be generated by default from the mth5, and intervals to exclude some data can be added as needed.

(b) is really just the case of considering pairs of tables like (a)

Question: To return a copy or modify in-place when querying. Need to decide on standards and syntax. Handling this in general is messy because every function needs to be modified. Maybe better to use a decorator that allows for df kwarg to be passed, and if it is not passed the modification is done in place. The user who doesn't want to modify in place can work with a clone.

**add\_columns\_for\_processing()** → None

Add columns to the dataframe used during processing.

Development Notes: - This was originally in pipelines. - Q: Should mth5\_objs be keyed by survey-station?  
- A: Yes, and ... since the KernelDataset dataframe will be iterated over, should probably write an iterator method. This can iterate over survey-station tuples for multiple station processing. - Currently the model of keeping all these data objects “live” in the df seems to work OK, but is not well suited to HPC or lazy processing. :param mth5\_objs: Keys are station\_id, values are MTH5 objects. :type mth5\_objs: dict,

**clone()**

Return a deep copy.

**clone\_dataframe()** → DataFrame

Return a deep copy of dataframe.

**close\_mth5s()** → None

Loop over all unique mth5\_objs in dataset df and make sure they are closed.+.

**property df**

Df function.

**drop\_runs\_shorter\_than(minimum\_duration: float, units: str | None = 's', inplace: bool | None = True)**  
→ DataFrame

Drop runs from df that are inconsequentially short

Development Notes: This needs to have duration refreshed before hand. :param inplace:

Defaults to True.

#### Parameters

- **minimum\_duration** (*float*) – The minimum allowed duration for a run (in units of units).
- **units** (*Optional[str]*, *optional*) – Placeholder to support units that are not seconds, defaults to “s”.

**from\_run\_summary**(*run\_summary: RunSummary*, *local\_station\_id: str | None = None*, *remote\_station\_id: str | None = None*, *sample\_rate: float | int | None = None*) → *None*

Initialize the dataframe from a run summary. :param sample\_rate:

Defaults to None.

#### Parameters

- **run\_summary** (*RunSummary*) – Summary of available data for processing from one or more stations.
- **local\_station\_id** (*Optional[Union[str, None]]*, *optional*) – Label of the station for which an estimate will be computed, defaults to None.
- **remote\_station\_id** (*Optional[Union[str, None]]*, *optional*) – Label of the remote reference station, defaults to None.

**get\_run\_object**(*index\_or\_row: int | Series*) → *Run*

Gets the run object associated with a row of the df

Development Notes: TODO: This appears to be unused except by get\_station\_metadata.

Delete or integrate if desired. - This has likely been deprecated by direct calls to `run_obj = row.mth5_obj.from_reference(row.run_hdf5_reference)` in pipelines..

#### Parameters

**index\_or\_row** (*Union[int, pd.Series]*)

#### Return run\_obj

The run associated with the row of the df.

#### Rtype run\_obj

`mt_metadata.timeseries.Run`

**get\_station\_metadata**(*local\_station\_id: str*) → *Station*

Returns the station metadata.

Development Notes: TODO: This appears to be unused. Was probably a precursor to the

`update_survey_metadata()` method. Delete if unused. If used fill out doc:

“Helper function for archiving the TF – returns an object we can use to populate station metadata in the \_\_\_\_”. :param local\_station\_id: The name of the local station. :type local\_station\_id: str :rtype: `mt_metadata.timeseries.Station`

**has\_local\_mth5()** → *bool*

Test if local mth5 exists.

**has\_remote\_mth5()** → *bool*

Test if remote mth5 exists.

**initialize\_dataframe\_for\_processing()** → None

Adds extra columns needed for processing to the dataframe.

Populates them with mth5 objects, run\_hdf5\_reference, and xr.Datasets.

Development Notes: Note #1: When assigning xarrays to dataframe cells, df dislikes xr.Dataset, so we convert to xr.DataArray before packing df

**Note #2: [OPTIMIZATION] By accessing the run\_ts and packing the “run\_dataarray” column of the df, we**

perform a non-lazy operation, and essentially forcing the entire decimation\_level=0 dataset to be loaded into memory. Seeking a lazy method to handle this maybe worthwhile. For example, using a df.apply() approach to initialize only one row at a time would allow us to generate the FCs one row at a time and never ingest more than one run of data at a time ...

**Note #3: Uncommenting the continue statement here is desireable, will speed things up, but**

is not yet tested. A nice test would be to have two stations, some runs having FCs built and others not having FCs built. What goes wrong is in update\_survey\_metadata. Need a way to get the survey metadata from a run, not a run\_ts if possible

**initialize\_mth5s(mode: str | None = 'r')**

Returns a dict of open mth5 objects, keyed by station\_id

A future version of this for multiple station processing may need nested dict with [survey\_id][station].  
:return mth5\_objs: Keyed by stations.

local station id : mth5.mth5.MTH5 remote station id: mth5.mth5.MTH5.

**Rtype mth5\_objs**

dict

**property input\_channels: list**

Get input channels from data frame. :return: Input channels (sources). :rtype: list of strings

**property is\_single\_station: bool**

Returns True if no RR station.

**property local\_df: DataFrame**

Split data frame to just the local station runs. :return: Local station runs. :rtype: pd.DataFrame

**property local\_mth5\_path: Path**

Local mth5 path. :return: Local station MTH5 path, a property extracted from the dataframe. :rtype: Path

**property local\_station\_id: str**

Local station id.

**property local\_survey\_id: str**

Return string label for local survey id.

**property local\_survey\_metadata: Survey**

Return survey metadata for local station.

**property mini\_summary: DataFrame**

Return a dataframe that fits in terminal.

**property mth5\_objs**

Mth5 objs. :return: Dictionary [station\_id: mth5\_obj]. :rtype: dict

**property num\_sample\_rates: int**

Returns the number of unique sample rates in the dataframe.

**property output\_channels: list**

Get input channels from data frame. :return: Input channels (sources). :rtype: list of strings

**property processing\_id: str**

Its difficult to come up with unique ids without crazy long names so this is a generic id of local-remote, the station metadata will have run information and the config parameters.

**property remote\_df: DataFrame**

Split data frame to just the local station runs. :return: Local station runs. :rtype: pd.DataFrame

**property remote\_mth5\_path: Path**

Remote mth5 path. :return: Remote station MTH5 path, a property extracted from the dataframe. :rtype: Path

**property remote\_station\_id: str**

Remote station id.

**restrict\_run\_intervals\_to\_simultaneous(df: DataFrame) → None**

For each run in local\_station\_id check if it has overlap with other runs

There is room for optimization here

Note that you can wind up splitting runs here. For example, in that case where local is running continuously, but remote is intermittent. Then the local run may break into several chunks.. :rtype: None

**property sample\_rate: float**

Returns the sample rate that of the data in the dataframe.

**select\_station\_runs(station\_runs\_dict: dict, keep\_or\_drop: bool, inplace: bool | None = True) → DataFrame**

Partition the rows of df based on the contents of station\_runs\_dict and return one of the two partitions (based on value of keep\_or\_drop).

dict -> {station: [{run, start, end}]}  
 For example {"mt01": ["0001", "0003"]} :param inplace:

Defaults to True.

**Parameters**

- **station\_runs\_dict (dict)** – Keys are string ids of the stations to keep Values are lists of string labels for run\_ids to keep.
- **keep\_or\_drop (bool)** – If “keep”: returns df with only the station-runs specified in station\_runs\_dict If “drop”: returns df with station\_runs\_dict excised.
- **overwrite (bool)** – If True, self.df is overwritten with the reduced dataframe.

**Return type**

pd.DataFrame

**classmethod set\_path(value: str | Path) → Path**

Set path.

**set\_run\_times**(*run\_time\_dict: dict, inplace: bool | None = True*)

Set run times from a dictionary formatted as {run\_id: {start, end}}}. :param run\_time\_dict: DESCRIPTION. :type run\_time\_dict: dict :param inplace: DESCRIPTION, defaults to True. :type inplace: Optional[bool], optional :return: DESCRIPTION. :rtype: TYPE

**update\_survey\_metadata**(*i: int, row: Series, run\_ts: RunTS*) → None

Wrangle survey\_metadata into kernel\_dataset.

Development Notes: - The survey metadata needs to be passed to TF before exporting data. - This was factored out of initialize\_dataframe\_for\_processing - TODO: It looks like we don't need to pass the whole run\_ts, just its metadata

There may be some performance implications to passing the whole object. Consider passing run\_ts.survey\_metadata, run\_ts.run\_metadata, run\_ts.station\_metadata only

### Parameters

- **i** (*int*) – This would be the index of row, if we were sure that the dataframe was cleanly indexed.
- **row** (*pd.Series*)
- **run\_ts** (*mth5.timeseries.run\_ts.RunTS*) – Mth5 object having the survey\_metadata.

### Return type

None

**class** mtpy.processing.RunSummary(*input\_dict: dict | None = None, df: DataFrame | None = None*)

Bases: object

Class to contain a run-summary table from one or more mth5s.

WIP: For the full MMT case this may need modification to a channel based summary.

**clone()**

2022-10-20: Cloning may be causing issues with extra instances of open h5 files ...

**property df: DataFrame**

Df function.

**drop\_no\_data\_rows()** → bool

Drops rows marked *has\_data* = False and resets the index of self.df.

**from\_mth5s**(*mth5\_list*) → list

Iterates over mth5s in list and creates one big dataframe summarizing the runs

**property mini\_summary: DataFrame**

Shows the dataframe with only a few columns for readability.

**property print\_mini\_summary: str**

Calls minisummary through logger so it is formatted.

**set\_sample\_rate**(*sample\_rate: float, inplace: bool = False*)

Set the sample rate so that the run summary represents all runs for a single sample rate.

### Parameters

- **sample\_rate** (*float*)
- **inplace** (*bool, optional*) – DESCRIPTION. By default, False.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**mtpy.utils package****Submodules****mtpy.utils.array2raster module****mtpy.utils.basemap\_tools module**

```
mtpy.utils.basemap_tools.add_basemap_frame(basemap, tick_interval=None, coastline_kwargs={},
states_kwargs={}, mlables=[False, False, False, True],
plabels=[True, False, False, False])
```

Add a standard map frame (lat/lon labels and tick marks, coastline and states) to basemap. :param basemap: :param tick\_interval: Tick interval in degrees, defaults to None. :param coastline\_kwargs: Dictionary containing arguments to pass into the drawcoastlines function, defaults to {} . :param states\_kwargs: Dictionary containing arguments to pass into the drawstates function, defaults to {} . :param mlables: Where to place meridian (longitude) labels on plot (list containing True/False for [left,right,top,bottom]), defaults to [False, False, False, True]. :param plabels: Where to place parallels (latitudes) labels on plot (list containing True/False for [left,right,top,bottom]), defaults to [True, False, False, False].

```
mtpy.utils.basemap_tools.compute_lonlat0_from_modem_data(stations_obj)
```

Compute lat0 and lon0 for creating a basemap, using data centre point in modem data file.

```
mtpy.utils.basemap_tools.compute_map_extent_from_modem_data(stations_obj, buffer=None,
buffer_factor=0.1)
```

Compute extent for a plot from data extent from ModEM data file. :param buffer\_factor:

Defaults to 0.1.

**Parameters**

- **stations\_obj**
- **data\_fn** – Full path to modem data file.
- **buffer** – Optional argument; buffer in latitude/longitude (if not provided,, defaults to None).

```
mtpy.utils.basemap_tools.compute_tick_interval_from_map_extent(lonMin, lonMax, latMin, latMax)
```

Estimate an even tick interval based on map extent based on some sensible options.

```
mtpy.utils.basemap_tools.get_latlon_extents_from_modem_data(stations_obj)
```

Get latlon extents from modem data.

```
mtpy.utils.basemap_tools.initialise_basemap(stations_obj, buffer=None, **basemap_kwargs)
```

Create a new basemap instance.

```
mtpy.utils.basemap_tools.plot_data(x, y, values, basemap=None, cbar=False, **param_dict)
```

Plot array data, either 1d or 2d. :param **\*\*param\_dict**: :param x: X position of points. :param y: Y position of points. :param values: Values to plot, if 1D, a scatter plot will be made, if 2D, a pcolormesh plot will be made. :param basemap: Supply a basemap, if None, data will be plotted on current axes, defaults to None. :param cbar: True/False, whether or not to show a colorbar, defaults to False.

## mtpy.utils.calculator module

mtpy/utils/calculator.py

Helper functions for standard calculations, e.g. error propagation

@UofA, 2013 (LK)

`mtpy.utils.calculator.centre_point(xarray, yarray)`

Get the centre point of arrays of x and y values.

`mtpy.utils.calculator.compute_determinant_error(z_array, z_err_array, method='theoretical', repeats=1000)`

Compute the error of the determinant of z using a stochastic method seed random z arrays with a normal distribution around the input array :param repeats:

Defaults to 1000.

### Parameters

- **z\_array** – Z (impedance) array containing real and imaginary values.
- **z\_err\_array** – Impedance error array containing real values, in MT we assume the real and imag errors are the same.
- **method** – Method to use, theoretical calculation or stochastic, defaults to “theoretical”.

### Returns

Error: array of real values with same shape as z\_err\_array representing the error in the determinant of Z.

### Returns

Error\_sqrt: array of real values with same shape as z\_err\_array representing the error in the (determinant of Z)\*\*0.5.

`mtpy.utils.calculator.get_period_list(period_min, period_max, periods_per_decade, include_outside_range=True)`

Get a list of values (e.g. periods), evenly spaced in log space and including values on multiples of 10 :return s: Numpy array containing list of values.

`mtpy.utils.calculator.get_rotation_matrix(angle, clockwise=False)`

get the rotation matrix for the proper rotation

### Parameters

- **angle** (*float*) – angle in degrees to rotate by
- **clockwise** (*bool*) – [ True | False ] if False counterclockwise rotation matrix is returned

`mtpy.utils.calculator.invertmatrix_incl_errors(inmatrix, inmatrix_error=None)`

Invertmatrix incl errors.

`mtpy.utils.calculator.make_log_increasing_array(z1_layer, target_depth, n_layers, increment_factor=0.999)`

Create depth array with log increasing cells, down to target depth, inputs are z1\_layer thickness, target depth, number of layers (n\_layers)

`mtpy.utils.calculator.multiplymatrices_incl_errors(inmatrix1, inmatrix2, inmatrix1_error=None, inmatrix2_error=None)`

Multiplymatrices incl errors.

**mtpy.utils.calculator.nearest\_index(val, array)**

Find the index of the nearest value in the array. :param val: The value to search for. :param array: The array to search in. :return: Index: integer describing position of nearest value in array.

**mtpy.utils.calculator.propagate\_error\_polar2rect(r, r\_error, phi, phi\_error)**

Find error estimations for the transformation from polar to cartesian coordinates.

Uncertainties in polar representation define a section of an annulus. Find the 4 corners of this section and additionally the outer boundary point, which is defined by phi = phi0, rho = rho0 + sigma rho. The cartesian “box” defining the uncertainties in x,y is the outer bound around the annulus section, defined by the four outermost points. So check the four corners as well as the outer boundary edge of the section to find the extrema in x and y. These give you the sigma\_x/y.

**mtpy.utils.calculator.propagate\_error\_rect2polar(x, x\_error, y, y\_error)**

Propagate error rect2polar.

**mtpy.utils.calculator.reorient\_data2D(x\_values, y\_values, x\_sensor\_angle=0, y\_sensor\_angle=90)**

Re-orient time series data of a sensor pair, which has not been in default (x=0, y=90) orientation.

Input: - x-values - Numpy array - y-values - Numpy array Note: same length for both! - If not, the shorter length is taken

Optional: - Angle of the x-sensor - measured in degrees, clockwise from North (0) - Angle of the y-sensor - measured in degrees, clockwise from North (0)

Output: - corrected x-values (North) - corrected y-values (East)

**mtpy.utils.calculator.rhophi2z(rho, phi, freq)**

Convert impedance-style information given in Rho/Phi format into complex valued Z.

Input: rho - 2x2 array (real) - in Ohm m phi - 2x2 array (real) - in degrees freq - scalar - frequency in Hz

Output: Z - 2x2 array (complex)

**mtpy.utils.calculator.rotate\_matrix\_errors(error, angle)**

Rotate errors of a matrix

**Parameters**

- **error** (*np.array*) – error matrix (2 x 2)
- **angle** (*float*) – rotation angle in degrees

**Returns**

propogated errors

**Return type**

None or np.array

**mtpy.utils.calculator.rotate\_matrix\_with\_errors(in\_matrix, angle, error=None, clockwise=True)**

Rotate a matrix including errors given an angle in degrees.

**Parameters**

- **in\_matrix** (*np.ndarray*) – A n x 2 x 2 matrix to rotate.
- **angle** (*float*) – Angle to rotate by assuming clockwise positive from 0 = north.
- **error** (*np.ndarray, optional*) – A n x 2 x 2 matrix of associated errors,, defaults to None.
- **clockwise** (*bool*) – rotate clockwise [True] or counter-clockwise [False]

**Raises**

**MTex** – If input array is incorrect.

**Returns**

Rotated matrix.

**Return type**

np.ndarray

**Returns**

Rotated matrix errors.

**Return type**

np.ndarray

`mtpy.utils.calculator.rotate_vector_with_errors(in_vector, angle, error=None, clockwise=True)`

Rotate a vector including errors given an angle in degrees.

**Parameters**

- **in\_vector**
- **in\_matrix (np.ndarray)** – A n x 1 x 2 vector to rotate.
- **angle (float)** – Angle to rotate by assuming clockwise positive from 0 = north.
- **error (np.ndarray, optional)** – A n x 1 x 2 vector of associated errors,, defaults to None.

**Raises**

**MTex** – If input array is incorrect.

**Returns**

Rotated vector.

**Return type**

np.ndarray

**Returns**

Rotated vector errors.

**Return type**

np.ndarray

`mtpy.utils.calculator.roundsf(number, sf)`

Round a number to a specified number of significant figures (sf).

`mtpy.utils.calculator.z_error2r_phi_error(z_real, z_imag, error)`

Error estimation from rectangular to polar coordinates.

By standard error propagation, relative error in resistivity is 2\*relative error in z amplitude.

Uncertainty in phase (in degrees) is computed by defining a circle around the z vector in the complex plane. The uncertainty is the absolute angle between the vector to (x,y) and the vector between the origin and the tangent to the circle. :return s: Tuple containing relative error in resistivity, absolute error in phase.

## [mtpy.utils.concatenate\\_input module](#)

**Description:**

This script collates data from raw data files in a folder, within a time-range provided by the user and outputs corresponding .EX, .EY, .EZ, .BX, .BY and .BZ files in an output folder.

## References:

CreationDate: 2017/10/23 Developer: [rakib.hassan@gar.gov.au](mailto:rakib.hassan@gar.gov.au)

## Revision History:

LastUpdate: 2017/10/23 RH

```
class mtpy.utils.concatenate_input.Data(dataPath, startDateTime='', endDateTime='')
```

Bases: object

**output**(*prefix*, *outputPath*)

Ouput function. :param prefix: Output file prefix. :param outputPath: Output folder.

## mtpy.utils.configfile module

Helper functions for the handling of configuration files (survey.cfg and BIRRP.cfg style).

@UofA, 2013 (LK)

```
mtpy.utils.configfile.read_configfile(filename)
```

Read a general config file and return the content as dictionary.

Config files without sections or only DEFAULT section -> return dictionary

Config files with sections -> return nested dictionary (main level keys are section heads)

Config files with sections as well as section-less entries -> return nested dictionary, which includes a top level 'DEFAULT' key

```
mtpy.utils.configfile.read_survey_configfile(filename)
```

Read in a survey configuration file and return a dictionary.

Input config file must contain station names as section headers!

The output dictionary keys are station names (capitalised), the values are (sub-)dictionaries. The configuration file must contain sections for all stations, each containing all mandatory keywords:

- latitude (deg)
- longitude (deg)
- elevation (in meters)
- sampling\_interval (in seconds)
- station\_type (MT, (Q)E, (Q)B)

Not mandatory, but recommended - declination (in degrees, positive to East) - this is set to '0.0', if omitted

Depending on the type of station the following entries are required.

E-field recorded:

- E\_logger\_type ('edl'/'elogger'/'qel')
- E\_logger\_gain (factor/gain-level)
- E\_instrument\_type ('electrodes'/'dipole')
- E\_instrument\_amplification (applied amplification factor)
- E\_Xaxis\_azimuth (degrees)
- E\_Xaxis\_length (in meters)
- E\_Yaxis\_azimuth (degrees)

- E\_Yaxis\_length (in meters)

B-field recorded: - B\_logger\_type ('edl'/'qel\_blogger') - B\_logger\_gain (factor/gain level) - B\_instrument\_type ('coil(s)', 'fluxgate') - B\_instrument\_amplification (applied amplification factor) - B\_Xaxis\_azimuth (degrees) - B\_Yaxis\_azimuth (degrees)

A global section can be used to include parameters for all stations. The name of the section must be one of:

global/main/default/general

`mtpy.utils.configfile.read_survey_txt_file(survey_file, delimiter=None)`

#### Read survey file and return a dictionary of dictionaries where the first

nested dictionary is keyed by the station name. Each station dictionary includes all the information input in the survey file with keywords verbatim as the headers in survey file, all lower case.

*Must be included in survey file ======  
key word description ======  
station station name lat(itude) latitude (decimal degrees is best) long(itude) longitude (decimal degrees is best) elev(ation) elevation (in meters) ex/E\_Xaxis\_length dipole length in north direction (in meters) ey/E\_Yaxis\_length dipole length in east direction (in meters) E\_Xaxis\_azimuth orientaion of Ex (degrees) E\_Yaxis\_azimuth orientaion of Ey (degrees)*

sampling\_interval sampling interval in seconds hx coil number in north direction for calibration hy coil number in east direction for calibration hz coil number in vertical direction for calibration date date of deployment notes any notes that might help later station\_type type of data collected (MT, E, B) declination declination in degrees (N = 0 and East = 90) ======  
=====

Information	on	E-field	data:	=====	key	word	description
E_logger_type	type of data logger used to record data	E_logger_gain	factor/gain level	E_instrument_type	type of electrodes used	E_instrument_amplification	applied amplification factor
E_Xaxis_azimuth	orientaion of Ex (degrees)	E_Xaxis_length	length of dipole for Ex (in meters)	E_Yaxis_azimuth	orientaion of Ey (degrees)	E_Yaxis_length	length of dipole for Ey (in meters)

#### Information on B-field data:

`survey_file` : string (full path to file)

`survey_lst`

[list] list of dictionaries with key words the same as the headers in survey file, all lower case

`mtpy.utils.configfile.write_config_from_survey_txt_file(survey_file, save_name=None, delimiter='\t')`

Write a survey configuration file from a survey txt file .

#### Arguments::

`survey_file`

[string] full path to survey text file. See `read_survey_txt_file` for the assumed header information.

`save_name`

[string] name to save file to. If `save_name` = None, then file saved as  
`os.path.join(os.path.dirname(survey_file,`

`os.path.basename(survey_file).cfg)`

**Outputs::****cfg\_fn**

[string] full path to saved config file

**mtpy.utils.configfile.write\_dict\_to\_configfile(dictionary, output\_filename)**

Write a dictionary into a configuration file.

The dictionary can contain pure key-value pairs as well as a level-1 nested dictionary. In the first case, the entries are stored in a ‘DEFAULT’ section. In the latter case, the dictionary keys are taken as section heads and the sub-dictionaries key-value pairs fill up the respective section

**mtpy.utils.convert\_modem\_data\_to\_geogrid module****mtpy.utils.edi\_folders module****Description:**

Find path to all the directories which contain a given type of files: .edi, .py .jpg, .pdf

**How to Run:**

```
python mtpy/utils/edi_folders.py . EDI python mtpy/utils/edi_folders.py /e/Data/ EDI 2 python
mtpy/utils/edi_folders.py /e/Data/ PY
```

CreationDate: 26/11/2017 Developer: fei.zhang@gov.au

**Revision History:**

LastUpdate: 26/11/2017 FZ started the first version

**class mtpy.utils.edi\_folders.EdiFolders(startDir, edifiles\_threshold=1, filetype='.edi')**

Bases: object

**find\_edi\_folders(aStartDir)**

Find directories containing the file of type self.filetype. :param aStartDir: The directory to start from.  
:return: A list of full path to folders of interest.

**get\_all\_edi\_files()**

Get all edi files.

**mtpy.utils.edi\_folders.recursive\_glob(dirname, ext='\*.edi')**

Under the dirname recursively find all files with extension ext.

Return a list of the full-path to the types of files of interest.

This function is useful to handle a nested directories of EDI files. :param dirname: A single dir OR a list of dirs.  
:param ext: Eg, “.edi”, “.xml”, defaults to “\*.edi”. :return: A list of path2files.

**mtpy.utils.estimate\_tf\_quality\_factor module**

Created on Wed Mar 6 09:43:07 2019

**Estimate Transfer Function Quality**

- based on simple statistics

@author: jpeacock

**class mtpy.utils.estimate\_tf\_quality\_factor.EMTFStats(z\_object, t\_object, \*\*kwargs)**

Bases: object

Class to estimate data quality of EM transfer functions. :param \*\*kwargs: :param t\_object: :param z\_object:  
:param tf\_dir: Transfer function directory. :type tf\_dir: string :param stat\_limits: Criteria for statistics based on  
a 0-5 rating scale. :type stat\_limits: dictionary

**compute\_statistics()**

Compute statistics of the transfer functions in a given directory.

Statistics are:

- one-lag autocorrelation coefficient, estimator for smoothness
- average of errors on components
- fit to a least-squares smooth curve
- normalized standard deviation of the first derivative, another smoothness estimator

**Parameters**

**tf\_dir** (*string*) – Path to directory of transfer functions.

**Return s**

Data frame of all the statistics estimated.

**Rtype s**

pandas.DataFrame

**estimate\_data\_quality(*stat\_df*)**

Convert the statistical estimates into the rating between 0-5 given a certain criteria.

**Note**

To change the criteria change self.stat\_limits

**Parameters**

- **stat\_df** (*pandas.DataFrame*) – Dataframe of the statistics.
- **stat\_fn** (*string*) – Name of .csv file of statistics.

**Return s**

A dataframe of the converted statistics.

**Rtype s**

pandas.DataFrame

**estimate\_quality\_factor(*weights*={'bad': 0.35, 'corr': 0.2, 'diff': 0.2, 'fit': 0.05, 'std': 0.2}, *round\_qf=False*)**

Convenience function doing all the steps to estimate quality factor.

**locate\_bad\_phase\_points(*phase*, *test*=5)**

Try to locate bad points to remove.

**locate\_bad\_res\_points(*res*)**

Try to locate bad points to remove.

**locate\_bad\_tipper\_points(*tipper*, *test*=0.2)**

Try to locate bad points to remove.

**summarize\_data\_quality(*quality\_df*, *weights*={'bad': 0.35, 'corr': 0.2, 'diff': 0.2, 'fit': 0.05, 'std': 0.2})**

Summarize the data quality into a single number for each station. :param quality\_df: Dataframe of the quality factors. :type quality\_df: pandas.DataFrame :param quality\_fn: Name of .csv file of quality factors. :type quality\_fn: string :return s: A dataframe of the summarized quality factors. :rtype s: pandas.DataFrame

## mtpy.utils.exceptions module

Specific exceptions for MTpy.

@UofA, 2013 (LK)

**exception mtpy.utils.exceptions.MTTimeError**

Bases: Exception

**exception mtpy.utils.exceptions.MTpyError\_EDI**

Bases: Exception

**exception mtpy.utils.exceptions.MTpyError\_PT**

Bases: Exception

**exception mtpy.utils.exceptions.MTpyError\_Tipper**

Bases: Exception

**exception mtpy.utils.exceptions.MTpyError\_Z**

Bases: Exception

**exception mtpy.utils.exceptions.MTpyError\_config\_file**

Bases: Exception

**exception mtpy.utils.exceptions.MTpyError\_edi\_file**

Bases: Exception

**exception mtpy.utils.exceptions.MTpyError\_file\_handling**

Bases: Exception

**exception mtpy.utils.exceptions.MTpyError\_float**

Bases: Exception

**exception mtpy.utils.exceptions.MTpyError\_input\_arguments**

Bases: Exception

**exception mtpy.utils.exceptions.MTpyError\_module\_import**

Bases: Exception

**exception mtpy.utils.exceptions.MTpyError\_occam**

Bases: Exception

**exception mtpy.utils.exceptions.MTpyError\_parameter\_number**

Bases: Exception

**exception mtpy.utils.exceptions.MTpyError\_processing**

Bases: Exception

**exception mtpy.utils.exceptions.MTpyError\_ts\_data**

Bases: Exception

**exception mtpy.utils.exceptions.MTpyError\_value**

Bases: Exception

## **mtpy.utils.filehandling module**

Helper functions for file handling.

The various functions deal with renaming, sorting, concatenation of time series, extraction of names and times from filenames, reading configuration files, ....

@UofA, 2013 (LK)

**mtpy.utils.filehandling.EDL\_get\_starttime\_fromfilename(filename)**

Return starttime of data file in epoch seconds.

Starting time is determined by the filename. This has to be of the form  
‘somthing/\*.\*.stationname.ddmmyyHHMMSS.??’

**mtpy.utils.filehandling.EDL\_get\_stationname\_fromfilename(filename)**

Edl get stationname fromfilename.

**mtpy.utils.filehandling.EDL\_make\_Nhour\_files(n\_hours, inputdir, sampling, stationname=None, outputdir=None)**

See ‘EDL\_make\_dayfiles’ for description and syntax.

Only difference: output files are blocks of (max) N hours, starting to count at midnight (00:00h) each day.

Conditions:

1. 24%N = 0
2. input data files start on the hour marks

Not working yet!!

**mtpy.utils.filehandling.EDL\_make\_dayfiles(inputdir, sampling, stationname=None, outputdir=None)**

Concatenate ascii time series to dayfiles (calendar day, UTC reference).

Data can be within a single directory or a list of directories. However, the files in the directory(ies) ‘inputdir’ have to be for one station only, and named with a 2 character suffix, defining the channel!

If the time series are interrupted/discontinuous at some point, a new file will be started after that point, where the file index ‘idx’ is increased by 1. If no stationname is given, the leading non-datetime characters in the first filename are used.

Files are named as ‘stationname\_samplingrate\_date\_idx.channel’ Stationname, channel, and sampling are written to a header line.

Output data consists of a single column float data array. The data are stored into one directory. If ‘outputdir’ is not specified, a subdirectory ‘dayfiles’ will be created within the current working directory.

Note: Midnight cannot be in the middle of a file, because only file starts are checked for a new day!!

**mtpy.utils.filehandling.get\_filename(fn, save\_path, fn\_basename)**

Get file name from inputs. :param fn: DESCRIPTION. :type fn: TYPE :param save\_path: DESCRIPTION. :type save\_path: TYPE :param fn\_basename: DESCRIPTION. :type fn\_basename: TYPE :return: DESCRIPTION. :rtype: TYPE

**mtpy.utils.filehandling.get\_pathlist(masterdir, search\_stringlist=None, search\_stringfile=None, start\_dict={}, split=' ', extension='', folder=False)**

Get a list of files or folders by searching on a string contained in search\_stringlist or alternatively search\_stringfile returns: dictionary containing search strings as keys and file/folder as values

masterdir - directory to search in search\_stringlist = list containing string identifiers for files or folders,

e.g. k0101 will work for edifile k0101.edi or folder k0101.

**search\_stringfile = alternative to search\_stringlist (need to provide one)**

will get search\_stringlist from a file, full path or make sure you are in the correct directory!

start\_dict = starting dictionary to append to, default is an empty dict split = if no exact match is found, search string will be split using split

character, useful when matching up edi's to inversion directories that both contain additional characters

extension = file extension, e.g. '.edi'

**mtpy.utils.filehandling.get\_sampling\_interval\_fromdatafile(filename, length=3600)**

Find sampling interval from data file.

Provide data file (purely numerical content) and total data length in seconds (default 3600). Data are read in by the Numpy 'loadtxt'-function, the length of the data array yields the sampling interval.

Lines beginning with # are ignored.

**mtpy.utils.filehandling.get\_ts\_header\_string(header\_dictionary)**

Return a MTpy time series data file header string from a dictionary.

**mtpy.utils.filehandling.make\_unique\_filename(infn)**

Make unique filename.

**mtpy.utils.filehandling.make\_unique\_folder(wd, basename='run')**

Make a folder that doesn't exist already.

**mtpy.utils.filehandling.read1columntext(textfile)**

Read a list from a one column text file.

**mtpy.utils.filehandling.read\_2c2\_file(filename)**

Read in BIRRP 2c2 coherence files and return 4 lists containing [period],[freq],[coh],[zcoh]. Note if any of the coherences are negative a value of 0 will be given to them.

**mtpy.utils.filehandling.read\_data\_header(fn\_raw)**

Deprecated!!! USE

read\_ts\_header

INSTEAD

Read the header line of MTpy TS data files.

**input:**

MTpy TS data file name

**output:**

list of header elements: stationname, channel, sampling rate, starttime first sample, starttime last sample, unit, lat, lon, elevation

**mtpy.utils.filehandling.read\_geotiff(filename, target\_epsg=None, bounds=None)**

Read in a geotiff. :param bounds:

Defaults to None.

**Parameters**

- **filename** (TYPE) – DESCRIPTION.
- **target\_epsg** (TYPE, optional) – DESCRIPTION, defaults to None.

### Returns

DESCRIPTION.

### Return type

TYPE

#### `mtpy.utils.filehandling.read_stationdatafile(textfile, read_duplicates=True)`

Read a space delimited file containing station info of any sort - 3 columns: station x, y, ... - to a dictionary - station:[x,y,...] textfile = full path to text file read\_duplicates = True/False - if stations are listed more than once do you

want to read all information or just the first occurrence, default True

example: import mtpy.utils.filehandling as fh stationdict = fh.read\_stationxyfile(textfile)

#### `mtpy.utils.filehandling.read_surface_ascii(ascii_fn)`

Read in surface which is ascii format () unlike original function, returns numpy array of lon, lat, elev (no projections)

The ascii format is assumed to be: ncols 2743 nrows 2019 xllcorner 111.791666666667 (lon of lower left) yllcorner -45.341666666667 (lat of lower left) cellsize 0.016666666667 NODATA\_value -9999 elevation data origin (0,0) is NW upper left. NW —————> E ||| S

#### `mtpy.utils.filehandling.read_ts_file(mtdatalist)`

Read an MTpy TS data file and provide the content as tuple:

(station, channel,samplingrate,t\_min,nsamples,unit,lat,lon,elev, data) If header information is incomplete, the tuple is filled up with 'None'.

#### `mtpy.utils.filehandling.read_ts_header(tsfile)`

Read in the header line from MTpy timeseries data files.

Return header as dictionary. Return empty dict, if no header line was found.

#### `mtpy.utils.filehandling.reorient_files(lo_files, configfile, lo_stations=None, outdir=None)`

Reorient files.

#### `mtpy.utils.filehandling.sort_folder_list(wkdir, order_file, indices=[0, 9999], delimiter="")`

Sort subfolders in wkdir according to order in order\_file

wkdir = working directory containing subfolders order = full path to text file containing order.

needs to contain a string to search on that is the same length for each item in the list

indices = indices to search on; default take the whole string

returns a list of directories, in order..

#### `mtpy.utils.filehandling.validate_save_file(savepath=None, savefile=None, basename=None, prioritise_savefile=False)`

Return savepath, savefile and basename, ensuring they are internally consistent and populating missing fields from the others or using defaults.

Prioritises savepath and basename. I.e. if savepath, savefile and basename are all valid but inconsistent, savefile will be updated to reflect savepath and basename :param prioritise\_savefile:

Defaults to False.

### Parameters

- **savepath** – Directory to save to, defaults to None.

- **savefile** – Full file path to save to, defaults to None.
- **basename** – Base file name to save to, defaults to None.

### `mtpy.utils.filehandling.validate_ts_file(tsfile)`

Validate MTpy timeseries (TS) data file Return Boolean value True/False .

### `mtpy.utils.filehandling.write_ts_file_from_tuple(outfile, ts_tuple, fmt='%.8e')`

Write an MTpy TS data file, where the content is provided as tuple:

(station, channel,samplingrate,t\_min,nsamples,unit,lat,lon,elev, data)

todo: needs tuple-validation.

## `mtpy.utils.gis_tools module`

### **GIS\_TOOLS**

This module contains tools to help project between coordinate systems. The module will first use GDAL if installed. If GDAL is not installed then pyproj is used. A test has been made for new versions of GDAL which swap the input lat and lon when using transferPoint, so the user should not have to worry about which version they have.

Main functions are:

- `project_point_ll2utm`
- `project_point_utm2ll`

These can take in a point or an array or list of points to project.

**latitude and longitude can be input as:**

- ‘DD:mm:ss.ms’
- ‘DD.decimal\_degrees’
- float(DD.decimal\_degrees)

Created on Fri Apr 14 14:47:48 2017 Revised: 5/2020 JP Revised: 10/2023 JP

@author: jrpeacock

### `exception mtpy.utils.gis_tools.GISError`

Bases: `Exception`

### `mtpy.utils.gis_tools.assert_elevation_value(elevation)`

Make sure elevation is a floating point number. :param elevation: Elevation as a float or string that can convert. :type elevation: float or str

### `mtpy.utils.gis_tools.assert_lat_value(latitude)`

Make sure the latitude value is in decimal degrees, if not change it.

And that the latitude is within -90 < lat > 90. :param latitude: Latitude in decimal degrees or other format. :type latitude: float or string

### `mtpy.utils.gis_tools.assert_lon_value(longitude)`

Make sure the longitude value is in decimal degrees, if not change it.

And that the longitude is within -180 < lon > 180. :param longitude: :param latitude: Longitude in decimal degrees or other format. :type longitude: float or string

### `mtpy.utils.gis_tools.assert_minutes(minutes)`

Assert minutes.

`mtpy.utils.gis_tools.assert_seconds(seconds)`

Assert seconds.

`mtpy.utils.gis_tools.convert_position_float2str(position)`

Convert position float to a string in the format of DD:MM:SS. :param position: Decimal degrees of latitude or longitude. :type position: float :return s: Latitude or longitude in format of DD:MM:SS.ms.

`mtpy.utils.gis_tools.convert_position_str2float(position_str)`

Convert a position string in the format of DD:MM:SS to decimal degrees. :param position\_str: :param position: Latitude or longitude om DD:MM:SS.ms. :type position: float :return s: Latitude or longitude as a float.

`mtpy.utils.gis_tools.project_point(x, y, old_epsg, new_epsg)`

Transform point to new epsg. :param x: DESCRIPTION. :type x: TYPE :param y: DESCRIPTION. :type y: TYPE :param old\_epsg: DESCRIPTION. :type old\_epsg: TYPE :param new\_epsg: DESCRIPTION. :type new\_epsg: TYPE :return: DESCRIPTION. :rtype: TYPE

`mtpy.utils.gis_tools.project_point_ll2utm(lat, lon, datum='WGS84', epsg=None)`

Project a point that is in latitude and longitude to the specified UTM coordinate system. :param lon: :param lat: :param latitude: Latitude in [ ‘DD:mm:ss.ms’ | ‘DD.decimal’ | float ]. :type latitude: [ string | float ] :param longitude: Longitude in [ ‘DD:mm:ss.ms’ | ‘DD.decimal’ | float ]. :type longitude: [ string | float ] :param datum: Well known datum, defaults to “WGS84”. :type datum: string, optional :param epsg: EPSG number defining projection

(see <http://spatialreference.org/ref/> for moreinfo) Overrides utm\_zone if both are provided, defaults to None.

### Returns

Project point(s) \* tuple is (easting, northing,utm\_zone) \* recarray has attributes (easting, northing, utm\_zone, elevation).

### Return type

tuple if a single point, np.recarray if multiple points

`mtpy.utils.gis_tools.project_point_utm2ll(easting, northing, utm_epsg, datum_epsg=4326)`

Project a point that is in UTM to the specified geographic coordinate system. :param datum\_epsg:

Defaults to 4326.

### Parameters

- **utm\_epsg**
- **easting** (*float*) – Easting in meters.
- **northing** (*float*) – Northing in meters.
- **datum** (*string*) – Well known datum.
- **utm\_zone** ([ *string* | *int* ]) – Utm\_zone {0-9}{0-9}{C-X} or {+, -}{0-9}{0-9}.
- **epsg** ([ *int* | *string* ]) – EPSG number defining projection (see <http://spatialreference.org/ref/> for moreinfo) Overrides utm\_zone if both are provided.

### Returns

Project point(s) \* tuple is (easting, northing,utm\_zone) \* recarray has attributes (easting, northing, utm\_zone, elevation).

### Return type

tuple if a single point, np.recarray if multiple points

```
mtpy.utils.gis_tools.validate_input_values(values, location_type=None)
```

Make sure the input values for lat, lon, easting, northing will be an numpy array with a float data type

can input a string as a comma separated list :param location\_type:

Defaults to None.

#### Parameters

**values** ([ float / string / list / numpy.ndarray ]) – Values to project, can be given as: \* float \* string of a single value or a comma separate string ‘34.2, 34.5’ \* list of floats or string \* numpy.ndarray.

#### Returns

Array of floats.

#### Return type

numpy.ndarray(dtype=float)

## mtpy.utils.matplotlib\_utils module

```
mtpy.utils.matplotlib_utils.gen_hist_bins(uniq_period_list)
```

Gen hist bins.

```
mtpy.utils.matplotlib_utils.get_next_fig_num()
```

Get next fig num.

## mtpy.utils.mtpy\_decorator module

```
class mtpy.utils.mtpy_decorator.deprecated(reason)
```

Bases: object

#### Description:

used to mark functions, methods and classes deprecated, and prints warning message when it called decrators based on <https://stackoverflow.com/a/40301488>

#### Usage:

todo: write usage

Author: YingzhiGou Date: 20/06/2017

## mtpy.utils.plot\_rms\_iterations module

```
mtpy.utils.plot_rms_iterations.concatenate_log_files(directory)
```

Any file of the pattern ‘\*.log’ will be included.

The files are

#### sorted alphanumerically and this is the order they will be

concatenated in. It is up to the user to ensure that the files are named correctly to achieve the desired order.  
:param directory: :type directory: str

```
mtpy.utils.plot_rms_iterations.plot(metric, values, x_start=0, x_end=None, x_interval=1, y_start=None, y_end=None, y_interval=None, fig_width=1900, fig_height=1200, dpi=100, minor_ticks=True)
```

Plot function.

```
mtpy.utils.plot_rms_iterations.read(logfile)
```

Get a sequence of values from a ModEM logfile.

Each type of value

present in the logfile is collected and ordered by iteration.

param logfile

param Returns

Dict of str, float: A dictionary containing lists of metric values.

## mtpy.utils.sensor\_orientation\_correction module

Created on Fri Oct 7 23:28:58 2022

@author: jpeacock

```
mtpy.utils.sensor_orientation_correction.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\_error** – Defaults to None.
- **Ey** – Defaults to 90.
- **Ex** – Defaults to 0.
- **By** – Defaults to 90.
- **Bx** – Defaults to 0.
- **Z\_prime** – Impedance tensor to be adjusted.

### Returns

Adjusted impedance tensor.

### Rtype

`np.ndarray(Z_prime.shape, dtype='complex')`

**Return s**

Impedance tensor standard deviation in default orientation.

**Rtype s**

np.ndarray(Z\_prime.shape, dtype='real')

**mtpy.utils.shapefiles module****Module contents**

Got rid of the GDAL check because have moved geographic operations to use pyproj. There are still some functions that use GDAL, but those are for raster and shapefile making. Therefore the check is not needed.

Created on Tue Sep 5 14:35:54 2023

@author: jpeacock

**Submodules****mtpy.mtpy\_globals module****Description:**

keep all the mtpy global params constants in this module.

Author: [fei.zhang@gov.au](mailto:fei.zhang@gov.au)

FZ Last Updated: 2017-12-04 JP (2021-01-18) updated to use Path and get relative path locations.

**Module contents****MTpy**

**class** `mtpy.MT(fn=None, impedance_units='mt', **kwargs)`

Bases: TF, [MTLocation](#)

Basic MT container to hold all information necessary for a MT station including the following parameters.

Impedance and Tipper element nomenclature is E/H therefore the first letter represents the output channels and the second letter represents the input channels.

For example for an input of Hx and an output of Ey the impedance tensor element is Zyx.

Coordinate reference frame of the transfer function is by default is NED

- x = North
- y = East
- z = + Down

The other option is ENU

- x = East
- y = North
- z = + Up

Other input options for the NED are:

- “+”
- “z+”
- “nez+”

- “ned”
- “exp(+ iomega t)”
- “exp(+iomega t)”
- None

And for ENU:

- “\_”
- “z-”
- “enz-”
- “enu”
- “exp(- iomega t)”
- “exp(-iomega t)”

### **property Tipper**

Mtpy.core.z.Tipper object to hold tipper information.

### **property Z**

Mtpy.core.z.Z object to hold impedance tensor.

### **add\_model\_error(*comp*=[], *z\_value*=5, *t\_value*=0.05, *periods*=None)**

Add error to a station’s components for given period range. :param periods:

Defaults to None.

#### **Parameters**

- **t\_value** – Defaults to 0.05.
- **z\_value** – Defaults to 5.
- **station** (*string or list of strings*) – Name of station(s) to add error to.
- **comp** – List of components to add data to, valid components are, defaults to [].

#### **Returns**

Data array with added errors.

#### **Return type**

np.ndarray

### **add\_white\_noise(*value*, *inplace*=True)**

Add white noise to the data, useful for synthetic tests. :param value: DESCRIPTION. :type value: TYPE :param inplace: DESCRIPTION, defaults to True. :type inplace: TYPE, optional :return: DESCRIPTION. :type: TYPE

### **clone\_empty()**

Copy metadata but not the transfer function estimates.

### **compute\_model\_t\_errors(*error\_value*=0.02, *error\_type*='absolute', *floor*=False)**

Compute mode errors based on the error type

percent    error\_value    \*    t    absolute    error\_value    ======  
=====. :param error\_value: DESCRIPTION02, defaults to 0.02. :type error\_value: TYPE, optional :param error\_type: DESCRIPTION, defaults to

“absolute”. :type error\_type: TYPE, optional :param floor: DESCRIPTION, defaults to False. :type floor: TYPE, optional :return: DESCRIPTION. :rtype: TYPE

**compute\_model\_z\_errors(error\_value=5, error\_type='geometric\_mean', floor=True)**

Compute mode errors based on the error type

```
egbert error_value * sqrt(Zxy * Zyx) geometric_mean error_value * sqrt(Zxy * Zyx) arithmetic_mean error_value * (Zxy + Zyx) / 2 mean_od error_value * (Zxy + Zyx) / 2 off_diagonals zxx_error == zxy_error, zyx_error == zyy_error median error_value * median(z) eigen error_value * mean(eigen(z)) percent error_value * z absolute error_value =====. :param error_value: DESCRIPTION, defaults to 5. :type error_value: TYPE, optional :param error_type: DESCRIPTION, defaults to “geometric_mean”. :type error_type: TYPE, optional :param floor: DESCRIPTION, defaults to True. :type floor: TYPE, optional :return: DESCRIPTION. :rtype: TYPE
```

**property coordinate\_reference\_frame**

**copy()**

Copy function.

**edit\_curve(method='default', tolerance=0.05)**

try to remove bad points in a scientific way.

**estimate\_tf\_quality(weights={'bad': 0.35, 'corr': 0.2, 'diff': 0.2, 'fit': 0.05, 'std': 0.2}, round\_qf=False)**

Estimate tranfer function quality factor 0-5, 5 being the best. :param weights: DESCRIPTION, defaults to

```
{ “bad”: 0.35, “corr”: 0.2, “diff”: 0.2, “std”: 0.2, “fit”: 0.05, }.
```

**Param**

DESCRIPTION.

**Type**

TYPE

**Returns**

DESCRIPTION.

**Return type**

TYPE

**property ex\_metadata**

EX metadata.

**property ey\_metadata**

EY metadata.

**find\_flipped\_phase()**

Identify if the off-diagonal components are flipped from traditional quadrants. xy should be in the 1st quadaran (0-90 deg) and yx should be in the 3rd quadrant (-180 to -90 deg) :return: A dictionary of components with a bool for flipped or not

if flipped return value is True.

**Return type**

dict

**flip\_phase**(*zxx=False*, *zxy=False*, *zyx=False*, *zyy=False*, *txz=False*, *tzy=False*, *inplace=False*)

Flip the phase of a station in case its plotting in the wrong quadrant. :param inplace:

Defaults to False.

#### Parameters

- **tzy** – Defaults to False.
- **txz** – Defaults to False.
- **station** (*string or list*) – Name(s) of station to flip phase.
- **station** – Station name or list of station names.
- **zxx** (*TYPE, optional*) – Z\_xx, defaults to False.
- **zxy** (*TYPE, optional*) – Z\_xy, defaults to False.
- **zyy** (*TYPE, optional*) – Z\_yx, defaults to False.
- **zyx** (*TYPE, optional*) – Z\_yy, defaults to False.
- **tx** (*TYPE, optional*) – T\_zx, defaults to False.
- **ty** (*TYPE, optional*) – T\_zy, defaults to False.

#### Returns

New\_data.

#### Return type

np.ndarray

#### Returns

New mt\_dict with components removed.

#### Return type

dictionary

**from\_dataframe**(*mt\_df*, *impedance\_units='mt'*)

Fill transfer function attributes from a dataframe for a single station. :param mt\_df: :param df: DESCRIPTION. :type df: TYPE :param impedance\_units: [“mt” [mV/km/nT] | “ohm” [Ohms] ] :type impedance\_units: str

**property hx\_metadata**

HX metadata.

**property hy\_metadata**

HY metadata.

**property hz\_metadata**

HZ metadata.

**property impedance\_units**

impedance units

**interpolate**(*new\_period*, *method='slinear'*, *bounds\_error=True*, *f\_type='period'*, *z\_log\_space=False*, *\*\*kwargs*)

Interpolate the impedance tensor onto different frequencies. :param z\_log\_space:

Defaults to False.

#### Parameters

- **new\_period** (*np.ndarray*) – A 1-d array of frequencies to interpolate on to. Must be within the bounds of the existing frequency range, anything outside and an error will occur.
- **method** (*string, optional*) – Method to interpolate by, defaults to “slinear”.
- **bounds\_error** (*boolean, optional*) – Check for if input frequencies are within the original frequencies, defaults to True.
- **f\_type** (*string, defaults to 'period', optional*) – Frequency type can be [ ‘frequency’ | ‘period’ ], defaults to “period”.
- **\*\*kwargs** – Key word arguments for *interp*.

**Raises**

**ValueError** – If input frequencies are out of bounds.

**Returns**

New MT object with interpolated values.

**Return type**

`mtpy.core.MT`

**plot\_depth\_of\_penetration(\*\*kwargs)**

Plot Depth of Penetration estimated from Niblett-Bostick estimation. :param **\*\*kwargs**: DESCRIPTION. :type **\*\*kwargs**: TYPE :return: DESCRIPTION. :rtype: TYPE

**plot\_mt\_response(\*\*kwargs)**

Returns a `mtpy.imaging.plotresponse.PlotResponse` object

**Plot Response**

```
>>> mt_obj = mt.MT(edi_file)
>>> pr = mt.plot_mt_response()
>>> # if you need more info on plot_mt_response
>>> help(pr).
```

**plot\_phase\_tensor(\*\*kwargs)**

Plot phase tensor. :return: DESCRIPTION. :rtype: TYPE

**property pt**

`Mtpy.analysis.pt.PhaseTensor` object to hold phase tensor.

**remove\_component(zxx=False, zxy=False, zyy=False, zyx=False, tzx=False, tzy=False, inplace=False)**

Remove a component for a given station(s). :param `inplace`:

Defaults to False.

**Parameters**

- **tzy** – Defaults to False.
- **tzx** – Defaults to False.
- **station** (*string or list*) – Station name or list of station names.
- **zxx** (*TYPE, optional*) –  $Z_{xx}$ , defaults to False.
- **zxy** (*TYPE, optional*) –  $Z_{xy}$ , defaults to False.
- **zyy** (*TYPE, optional*) –  $Z_{yx}$ , defaults to False.

- **zyx** (*TYPE, optional*) – Z\_yy, defaults to False.
- **tx** (*TYPE, optional*) – T\_zx, defaults to False.
- **ty** (*TYPE, optional*) – T\_zy, defaults to False.

**Returns**

New data array with components removed.

**Return type**

np.ndarray

**Returns**

New mt\_dict with components removed.

**Return type**

dictionary

**remove\_distortion(*n\_frequencies=None, comp='det', only\_2d=False, inplace=False*)**

Remove distortion following Bibby et al. [2005]. :param inplace:

Defaults to False.

**Parameters**

- **only\_2d** – Defaults to False.
- **comp** – Defaults to “det”.
- **n\_frequencies** (*int, optional*) – Number of frequencies to look for distortion from the highest frequency, defaults to None.

**Return s**

Distortion matrix.

**Rtype s**

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

**Return s**

Z with distortion removed.

**Rtype s**

mtpy.core.z.Z

**remove\_static\_shift(*ss\_x=1.0, ss\_y=1.0, inplace=False*)**

Remove static shift from the apparent resistivity

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

- $Z = S * Z_0$

therefore the correct Z will be :

- $Z_0 = S^{-1} * Z$ .

**Parameters**

- **inplace** – Defaults to False.
- **ss\_x** (*float, optional*) – Correction factor for x component, defaults to 1.0.
- **ss\_y** (*float, optional*) – Correction factor for y component, defaults to 1.0.

**Returns**

New Z object with static shift removed.

**Rtype**

mtpy.core.z.Z

**rotate(theta\_r, inplace=True)**

Rotate the data in degrees assuming North is 0 measuring clockwise positive to East as 90.

**Parameters**

- **theta\_r** (*float*) – rotation angle to rotate by in degrees.
- **inplace** (*bool, optional*) – rotate all transfer function in place, defaults to True.

**Returns**

if inplace is False, returns a new MT object.

**Return type**

MT object

**property rotation\_angle**

Rotation angle in degrees from north. In the coordinate reference frame

**property rrhx\_metadata**

RRHX metadata.

**property rrhy\_metadata**

RRHY metadata.

**to\_dataframe(utm\_crs=None, cols=None, impedance\_units='mt')**

Create a dataframe from the transfer function for use with plotting and modeling. :param cols:

Defaults to None.

**Parameters**

- **utm\_crs** (*eter*) – Defaults to None.
- **utm\_crs** – The utm zone to project station to, could be a name, pyproj.CRS, EPSG number, or anything that pyproj.CRS can intake.
- **impedance\_units** (*str*) – [“mt” [mV/km/nT] | “ohm” [Ohms] ]

**to\_occam1d(data\_filename=None, mode='det')**

Write an Occam1DData data file. :param data\_filename: Path to write file, if None returns Occam1DData

object, defaults to None.

**Parameters**

**mode** (*string, optional*) – [ ‘te’, ‘tm’, ‘det’, ‘tez’, ‘tmz’, ‘detz’], defaults to “det”.

**Returns**

Occam1DData object.

**Return type**

*mtpy.modeling.occam1d.Occam1DData*

**to\_simpeg\_1d**(*mode='det'*, *\*\*kwargs*)  
Helper method to run a 1D inversion using Simpeg  
default is smooth parameters

#### To run sharp inversion

```
>>> mt_object.to_simpeg_1d({"p_s": 2, "p_z": 0, "use_irrls": True})
```

#### To run sharp inversion adn compact

```
>>> mt_object.to_simpeg_1d({"p_s": 0, "p_z": 0, "use_irrls": True}).
:param mode:
 Defaults to "det".
:param **kwargs: DESCRIPTION.
:type **kwargs: TYPE
:return: DESCRIPTION.
:rtype: TYPE
```

**class** `mtpy.MTCollection`(*working\_directory=None*)

Bases: object

Collection of transfer functions

The main working variable is *MTCollection.dataframe* which is a property that returns either the *master\_dataframe* that contains all the TF's in the MTH5 file, or the *working\_dataframe* which is a dataframe that has been queried in some way. Therefore all the user has to do is set the working directory as a subset of the *master\_dataframe*

#### Example

```
>>> mc = MTCollection()
>>> mc.open_collection(filename="path/to/example/mth5.h5")
>>> mc.working_dataframe = mc.master_dataframe.iloc[0:5].
```

**add\_tf**(*transfer\_function*, *new\_survey=None*, *tf\_id\_extra=None*)

Transfer\_function could be a transfer function object, a file name, a list of either.

#### Parameters

- **transfer\_function** (*list*, *tuple*, *array*, *MTData*, *MT*) – Transfer function object.
- **new\_survey** (*str*, *optional*) – New survey name, defaults to None.
- **tf\_id\_extra** (*string*, *optional*) – Additional text onto existing ‘tf\_id’, defaults to None.

#### Returns

DESCRIPTION.

#### Return type

TYPE

**apply\_bbox**(*lon\_min: float*, *lon\_max: float*, *lat\_min: float*, *lat\_max: float*) → None

Sets self.working\_dataframe to only stations within bounding box.

#### Parameters

- **lon\_min** (*float*) – Minimum longitude.

- **lon\_max** (*float*) – Maximum longitude.
- **lat\_min** (*float*) – Minimum latitude.
- **lat\_max** (*float*) – Maximum longitude.

**average\_stations**(*cell\_size\_m*, *bounding\_box*=None, *count*=1, *n\_periods*=48, *new\_file*=True)

Average nearby stations to make it easier to invert.

#### Parameters

- **new\_file** – Defaults to True.
- **n\_periods** – Defaults to 48.
- **count** – Defaults to 1.
- **cell\_size\_m** (*float*) – size of square to look in for nearby stations
- **bounding\_box** (*list, optional*) – bounding box [lon\_min, lon\_max, lat\_min, lat\_max], defaults to None.

**check\_for\_duplicates**(*locate*='location', *sig\_figs*=6)

Check for duplicate station locations in a MT DataFrame.

#### Parameters

- **sig\_figs** – Defaults to 6.
- **locate** – Defaults to “location”.
- **dataframe** (*TYPE*) – DESCRIPTION.

#### Returns

DESCRIPTION.

#### Return type

TYPE

**close\_collection()**

Close mth5.

#### Returns

DESCRIPTION.

#### Return type

TYPE

**property dataframe**

This property returns the working dataframe or master dataframe if the working dataframe is None.

#### Returns

DESCRIPTION.

#### Return type

TYPE

**from\_mt\_data**(*mt\_data*, *new\_survey*=None, *tf\_id\_extra*=None)

Add data from a MTData object to an MTH5 collection.

Can use ‘new\_survey’ to create a new survey to load to.

Can use ‘tf\_id\_extra’ to add a string onto the existing ‘tf\_id’, useful if data have been edited or manipulated in some way. For example could set ‘tf\_id\_extra’ = ‘rotated’ for rotated data. This will help you organize the tf’s for each station.

**Parameters**

- **mt\_data** (`mtpy.core.mt_data.MTData`) – MTData object.
- **new\_survey** (`str, optional`) – New survey name, defaults to None.
- **tf\_id\_extra** (`string, optional`) – Additional text onto existing ‘tf\_id’,, defaults to None.

**Raises**

`IOError` – If an MTH5 is not writable raises.

**get\_tf**(*tf\_id*, *survey*=None)

Get transfer function.

**Parameters**

- **survey** – Defaults to None.
- **tf\_id** (`TYPE`) – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**has\_data()**

Has data.

**static make\_file\_list**(*mt\_path*, *file\_types*=['edi'])

Get a list of MT file from a given path.

**Parameters**

- **file\_types** – Defaults to ['edi'].
- **mt\_path** – Full path to where the MT transfer functions are stored.

**property master\_dataframe**

This is the full summary of all transfer functions in the MTH5 file. It is a property because if a user adds TF’s then the master\_df will be automatically updated. the transformation is quick for now.

**property mth5\_filename**

Mth5 filename.

**open\_collection**(*filename*=None, *basename*=None, *working\_directory*=None, *mode*='a', \*\**kwargs*)

Initialize an mth5.

**Parameters**

- **mode** – Defaults to “a”.
- **filename** – Defaults to None.
- **basename** (`TYPE, optional`) – DESCRIPTION, defaults to None.
- **working\_directory** (`TYPE, optional`) – DESCRIPTION, defaults to None.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**plot\_mt\_response**(*tf\_id*, *survey=None*, *\*\*kwargs*)

Plot mt response.

#### Parameters

- **survey** – Defaults to None.
- **tf\_id** (*TYPE*) – DESCRIPTION.
- **\*\*kwargs** – DESCRIPTION.

#### Returns

DESCRIPTION.

#### Return type

*TYPE*

**plot\_penetration\_depth\_1d**(*tf\_id*, *survey=None*, *\*\*kwargs*)

Plot 1D penetration depth based on the Niblett-Bostick transformation

Note that data is rotated to estimated strike previous to estimation and strike angles are interpreted for data points that are 3D.

#### See also

[mtpy.analysis.niblettbostick.calculate\\_depth\\_of\\_investigation](#).

#### Parameters

- **survey** – Defaults to None.
- **tf\_id**
- **tf\_object** (*TYPE*) – DESCRIPTION.
- **\*\*kwargs** – DESCRIPTION.

#### Returns

DESCRIPTION.

#### Return type

*TYPE*

**plot\_penetration\_depth\_map**(*mt\_data=None*, *\*\*kwargs*)

Plot Penetration depth in map view for a single period

#### See also

[mtpy.imaging.PlotPenetrationDepthMap](#).

#### Parameters

- **\*\*kwargs** –
- **mt\_data** (*TYPE, optional*) – DESCRIPTION, defaults to None.

#### Returns

DESCRIPTION.

**Return type**

TYPE

**plot\_phase\_tensor**(*tf\_id*, *survey=None*, *\*\*kwargs*)

Plot phase tensor elements.

**Parameters**

- **survey** – Defaults to None.
- **tf\_id** (TYPE) – DESCRIPTION.
- **\*\*kwargs** – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**plot\_phase\_tensor\_map**(*mt\_data=None*, *\*\*kwargs*)

Plot Phase tensor maps for transfer functions in the working\_dataframe

 See also[mtpy.imaging.PlotPhaseTensorMaps](#).**Parameters**

- **mt\_data** – Defaults to None.
- **\*\*kwargs** – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**plot\_phase\_tensor\_pseudosection**(*mt\_data=None*, *\*\*kwargs*)

Plot a pseudo section of phase tensor ellipses and induction vectors if specified

 See also[mtpy.imaging.PlotPhaseTensorPseudosection](#)**Parameters**

- **mt\_data** – Defaults to None.
- **\*\*kwargs** – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**plot\_residual\_phase\_tensor**(*mt\_data\_01*, *mt\_data\_02*, *plot\_type='map'*, *\*\*kwargs*)

Plot residual phase tensor.

#### Parameters

- **mt\_data\_01** (*TYPE*) – DESCRIPTION.
- **mt\_data\_02** (*TYPE*) – DESCRIPTION.
- **plot\_type** (*TYPE, optional*) – DESCRIPTION, defaults to “map”.
- **\*\*kwargs** – DESCRIPTION.

#### Returns

DESCRIPTION.

#### Return type

*TYPE*

**plot\_resistivity\_phase\_maps**(*mt\_data=None*, *\*\*kwargs*)

Plot apparent resistivity and/or impedance phase maps from the working dataframe

#### See also

[\*mtpy.imaging.PlotResPhaseMaps\*](#)

#### Parameters

- **mt\_data** – Defaults to None.
- **\*\*kwargs** – DESCRIPTION.

#### Returns

DESCRIPTION.

#### Return type

*TYPE*

**plot\_resistivity\_phase\_pseudosections**(*mt\_data=None*, *\*\*kwargs*)

Plot resistivity and phase in a pseudosection along a profile line.

#### Parameters

- **mt\_data** (*TYPE, optional*) – DESCRIPTION, defaults to None.
- **\*\*kwargs** – DESCRIPTION.

#### Returns

DESCRIPTION.

#### Return type

*TYPE*

**plot\_stations**(*map\_epsg=4326*, *bounding\_box=None*, *\*\*kwargs*)

Plot stations.

#### Parameters

- **bounding\_box** – Defaults to None.
- **map\_epsg** – Defaults to 4326.
- **\*\*kwargs** – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**plot\_strike**(*mt\_data=None*, *\*\*kwargs*)

Plot strike angle

 **See also***mtpy.imaging.PlotStrike*.**to\_geo\_df**(*bounding\_box=None*, *epsg=4326*)

Make a geopandas dataframe for easier GIS manipulation.

**to\_mt\_data**(*bounding\_box=None*, *\*\*kwargs*)

Get a list of transfer functions.

**Parameters**

- **\*\*kwargs** –
- **tf\_ids** (TYPE, optional) – DESCRIPTION, defaults to None.
- **bounding\_box** (TYPE, optional) – DESCRIPTION, defaults to None.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**to\_shp**(*filename*, *bounding\_box=None*, *epsg=4326*)

Create a shape file of station locations in the given EPSG number

**Parameters**

- **filename** (str) – filename to save the shape file to.
- **bounding\_box** (list, optional) – bounding box [lon\_min, lon\_max, lat\_min, lat\_max], defaults to None.
- **epsg** (int, optional) – EPSG number to write shape file to, defaults to 4326.

**Returns**

dataframe.

**Return type**

geopandas.DataFrame

**property working\_directory**

Working directory.

**class mtpy.MTData**(*mt\_list=None*, *\*\*kwargs*)Bases: *OrderedDict*, *MTStations*Collection of MT objects as an *OrderedDict* where keys are formatted as *survey\_id.station\_id*. Has all functionality of an *OrderedDict* for example can iterate of *.keys()*, *.values()* or *.items*. Values are a list of MT objects.Inherits *mtpyt.core.MTStations* to deal with geographic locations of stations.

Is not optimized yet for speed, works fine for smaller surveys, but for large can be slow. Might try using a dataframe as the base.

```
add_station(mt_object, survey=None, compute_relative_location=True, interpolate_periods=None,
 compute_model_error=False)
```

Add a MT object.

#### Parameters

- **compute\_model\_error** – Defaults to False.
- **mt\_object** (`mtpy.MT`) – MT object for a single station.
- **survey** (`str, optional`) – New survey name, defaults to None.
- **compute\_relative\_location** (`bool, optional`) – Compute relative location, can be slow if adding single stations in a loop. If looping over station set to False and compute at the end, defaults to True.
- **interpolate\_periods** (`np.array, optional`) – Periods to interpolate onto, defaults to None.

```
add_tf(tf, **kwargs)
```

Add a MT object.

#### Parameters

- **\*\*kwargs** –
- **tf**
- **mt\_object** (`mtpy.MT`) – MT object for a single station.
- **survey** (`str, optional`) – New survey name, defaults to None.
- **compute\_relative\_location** (`bool, optional`) – Compute relative location, can be slow if adding single stations in a loop. If looping over station set to False and compute at the end, defaults to True.
- **interpolate\_periods** (`np.array, optional`) – Periods to interpolate onto, defaults to None.

```
add_white_noise(value, inplace=True)
```

Add white noise to the data, useful for synthetic tests.

#### Parameters

- **value** (`TYPE`) – DESCRIPTION.
- **inplace** (`TYPE, optional`) – DESCRIPTION, defaults to True.

#### Returns

DESCRIPTION.

#### Return type

TYPE

```
apply_bounding_box(lon_min, lon_max, lat_min, lat_max)
```

Apply a bounding box.

#### Parameters

- **lon\_min** (`TYPE`) – DESCRIPTION.
- **lon\_max** (`TYPE`) – DESCRIPTION.

- **lat\_min** (TYPE) – DESCRIPTION.
- **lat\_max** (TYPE) – DESCRIPTION.

**Returns**  
DESCRIPTION.

**Return type**  
TYPE

### clone\_empty()

Return a copy of MTData excluding MT objects.

**Returns**  
Copy of MTData object excluding MT objects.

**Return type**  
*mtpy.MTData*

**compute\_model\_errors**(*z\_error\_value=None*, *z\_error\_type=None*, *z\_floor=None*, *t\_error\_value=None*, *t\_error\_type=None*, *t\_floor=None*)

Compute mode errors based on the error type

key	definition
egbert	error_value * sqrt(Zxy * Zyx)
geometric_mean	error_value * sqrt(Zxy * Zyx)
arithmetic_mean	error_value * (Zxy + Zyx) / 2
mean_od	error_value * (Zxy + Zyx) / 2
off_diagonals	zxx_err == zxy_err, zyx_err == zyy_err
median	error_value * median(z)
eigen	error_value * mean(eigen(z))
percent	error_value * z
absolute	error_value

### Parameters

- **z\_error\_value** (TYPE, optional) – DESCRIPTION, defaults to None.
- **z\_error\_type** (TYPE, optional) – DESCRIPTION, defaults to None.
- **z\_floor** (TYPE, optional) – DESCRIPTION, defaults to None.
- **t\_error\_value** (TYPE, optional) – DESCRIPTION02, defaults to None.
- **t\_error\_type** (TYPE, optional) – DESCRIPTION, defaults to None.
- **t\_floor** (TYPE, optional) – DESCRIPTION, defaults to None.

**Param**  
DESCRIPTION.

**Type**  
TYPE

**Returns**  
DESCRIPTION.

**Return type**  
TYPE

**property coordinate\_reference\_frame**

coordinate reference frame ned or enu

**copy()**

Deep copy of original MTData object.

**Parameters**

**memo** (TYPE) – DESCRIPTION.

**Returns**

Deep copy of original MTData.

**Return type**

*mtpy.MTData*

**estimate\_spatial\_static\_shift(station\_key, radius, period\_min, period\_max, radius\_units='m', shift\_tolerance=0.15)**

Estimate static shift for a station by estimating the median resistivity values for nearby stations within a radius given. Can set the period range to estimate the resistivity values.

**Parameters**

- **shift\_tolerance** – Defaults to 0.15.
- **radius\_units** – Defaults to “m”.
- **station\_key** (TYPE) – DESCRIPTION.
- **radius** (TYPE) – DESCRIPTION.
- **period\_min** (TYPE) – DESCRIPTION.
- **period\_max** (TYPE) – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**estimate\_starting\_rho()**

Estimate starting resistivity from the data.

Creates a plot of the mean and median apparent resistivity values.

**Returns**

Array of the median rho per period.

**Return type**

np.ndarray(n\_periods)

**Returns**

Array of the mean rho per period.

**Return type**

np.ndarray(n\_periods)

**from\_dataframe(df, impedance\_units='mt')**

Create an dictionary of MT objects from a dataframe.

**Parameters**

- **df** (*pandas.DataFrame*) – Dataframe of mt data.
- **impedance\_units** (str) – [ “mt” [mV/km/nT] | “ohm” [Ohms] ]

**from\_modem**(*data\_filename*, *survey*='data', \*\**kwargs*)

Read in a modem data file

**Parameters**

- **survey** – Defaults to “data”.
- **data\_filename** (*TYPE*) – DESCRIPTION.
- **\*\*kwargs** – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

*TYPE*

**from\_modem\_data**(*data\_filename*, *survey*='data', \*\**kwargs*)

From modem data.

**Parameters**

- **survey** – Defaults to “data”.
- **data\_filename** (*TYPE*) – DESCRIPTION.
- **file\_type** (*TYPE, optional*) – DESCRIPTION, defaults to “data”.
- **\*\*kwargs** – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

*TYPE*

**from\_mt\_dataframe**(*mt\_df*, *impedance\_units*='mt')

Create an dictionary of MT objects from a dataframe.

**Parameters**

- **mt\_df**
- **df** (*MTDataFrame*) – Dataframe of mt data.
- **impedance\_units** (*str*) – [ “mt” [mV/km/nT] | “ohm” [Ohms] ]

**from\_occam2d**(*data\_filename*, *file\_type*='data', \*\**kwargs*)

Read in occam data from a 2D data file \*.dat

**Read data file and plot**

```
>>> from mtpy import MTData
>>> md = MTData()
>>> md.from_occam2d_data(f"/path/to/data/file.dat")
>>> plot_stations = md.plot_stations(model_locations=True)
```

**Read response file**

```
>>> md.from_occam2d_data(f"/path/to/response/file.dat")
```

**Note**

When reading in a response file the survey will be called model. So now you can have the data and model response in the same object..

**from\_occam2d\_data(data\_filename, file\_type='data', \*\*kwargs)**

From occam2d data.

**get\_nearby\_stations(station\_key, radius, radius\_units='m')**

Get stations close to a given station.

**Parameters**

- **radius\_units** – Defaults to “m”.
- **station\_key** (TYPE) – DESCRIPTION.
- **radius** (TYPE) – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**get\_periods()**

Get all unique periods.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**get\_profile(x1, y1, x2, y2, radius)**

Get stations along a profile line given the (x1, y1) and (x2, y2) coordinates within a given radius (in meters).

These can be in (longitude, latitude) or (easting, northing). The calculation is done in UTM, therefore a UTM CRS must be input

**Parameters**

- **x1** (TYPE) – DESCRIPTION.
- **y1** (TYPE) – DESCRIPTION.
- **x2** (TYPE) – DESCRIPTION.
- **y2** (TYPE) – DESCRIPTION.
- **radius** (TYPE) – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**get\_station(station\_id=None, survey\_id=None, station\_key=None)**

If ‘station\_key’ is None, tries to find key from *station\_id* and ‘survey\_id’ using MTData.\_get\_station\_key()

**Parameters**

- **station\_key** (*str, optional*) – Full station key {survey\_id}.{station\_id}., defaults to None.
- **station\_id** (*str, optional*) – Station ID, defaults to None.
- **survey\_id** (*str, optional*) – Survey ID, defaults to None.

**Raises**

**KeyError** – If cannot find station\_key.

**Returns**

MT object.

**Return type**

*mtpy.MT*

**get\_subset(station\_list)**

Get a subset of the data from a list of stations, could be station\_id or station\_keys. Safest to use keys {survey}.{station}

**Parameters**

**station\_list** (*list*) – List of station keys as {survey\_id}.{station\_id}.

**Returns**

Returns just those stations within station\_list.

**Return type**

*mtpy.MTData*

**get\_survey(survey\_id)**

Get all MT objects that belong to the ‘survey\_id’ from the data set.

**Parameters**

**survey\_id** (*str*) – Survey ID.

**Returns**

MTData object including only those with the desired ‘survey\_id’.

**Return type**

*mtpy.MTData*

**property impedance\_units**

impedance units

**interpolate(new\_periods, f\_type='period', inplace=True, bounds\_error=True, \*\*kwargs)**

Interpolate onto common period range

kwargs include

- method
- bounds\_error
- z\_log\_space
- na\_method
- extrapolate

**Parameters**

- **new\_periods** (*TYPE*) – DESCRIPTION
- **inplace** – Defaults to True.

- **new\_periods** – DESCRIPTION.
- **f\_type** (*string, defaults to 'period', optional*) – Frequency type can be [ ‘frequency’ | ‘period’ ], defaults to “period”.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**property mt\_list**

Mt list.

**Returns**

List of MT objects.

**Return type**

list

**property n\_stations**

Number of stations in MT data.

**plot\_mt\_response**(*station\_key=None, station\_id=None, survey\_id=None, \*\*kwargs*)

Plot mt response.

**Parameters**

- **survey\_id** – Defaults to None.
- **station\_id** – Defaults to None.
- **station\_key** – Defaults to None.
- **tf\_id** (TYPE) – DESCRIPTION.
- **\*\*kwargs** – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**plot\_penetration\_depth\_1d**(*station\_key=None, station\_id=None, survey\_id=None, \*\*kwargs*)

Plot 1D penetration depth based on the Niblett-Bostick transformation

Note that data is rotated to estimated strike previous to estimation and strike angles are interpreted for data points that are 3D.

 **See also**
[mtpy.analysis.niblettbostick.calculate\\_depth\\_of\\_investigation](#).
**Parameters**

- **survey\_id** – Defaults to None.
- **station\_id** – Defaults to None.
- **station\_key** – Defaults to None.

- **tf\_object** (*TYPE*) – DESCRIPTION.
- **\*\*kwargs** – DESCRIPTION.

**Returns**  
DESCRIPTION.

**Return type**  
TYPE

### `plot_penetration_depth_map(**kwargs)`

Plot Penetration depth in map view for a single period

#### See also

`mtpy.imaging.PlotPenetrationDepthMap.`

#### Parameters

- **\*\*kwargs** –
- **mt\_data** (*TYPE*) – DESCRIPTION.

**Returns**  
DESCRIPTION.

**Return type**  
TYPE

### `plot_phase_tensor(station_key=None, station_id=None, survey_id=None, **kwargs)`

Plot phase tensor elements.

#### Parameters

- **survey\_id** – Defaults to None.
- **station\_id** – Defaults to None.
- **station\_key** – Defaults to None.
- **tf\_id** (*TYPE*) – DESCRIPTION.
- **\*\*kwargs** – DESCRIPTION.

**Returns**  
DESCRIPTION.

**Return type**  
TYPE

### `plot_phase_tensor_map(**kwargs)`

Plot Phase tensor maps for transfer functions in the working\_dataframe

#### See also

`mtpy.imaging.PlotPhaseTensorMaps.`

#### Parameters

- **\*\*kwargs** – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**plot\_phase\_tensor\_pseudosection**(*mt\_data=None*, *\*\*kwargs*)

Plot a pseudo section of phase tensor ellipses and induction vectors if specified

 **See also**[mtpy.imaging.PlotPhaseTensorPseudosection](#)**Parameters**

- ***mt\_data*** – Defaults to None.
- ***\*\*kwargs*** – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**plot\_residual\_phase\_tensor\_maps**(*survey\_01*, *survey\_02*, *\*\*kwargs*)

Plot residual phase tensor maps.

**Parameters**

- ***\*\*kwargs*** –
- ***survey\_01* (TYPE)** – DESCRIPTION.
- ***survey\_02* (TYPE)** – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**plot\_resistivity\_phase\_maps**(*\*\*kwargs*)

Plot apparent resistivity and/or impedance phase maps from the working dataframe

 **See also**[mtpy.imaging.PlotResPhaseMaps](#)**Parameters*****\*\*kwargs*** – DESCRIPTION.**Returns**

DESCRIPTION.

**Return type**

TYPE

**plot\_resistivity\_phase\_pseudosections(\*\*kwargs)**

Plot resistivity and phase in a pseudosection along a profile line.

**Parameters**

- **mt\_data** (*TYPE*, *optional*) – DESCRIPTION, defaults to None.
- **\*\*kwargs** – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

*TYPE*

**plot\_stations(map\_epsg=4326, bounding\_box=None, model\_locations=False, \*\*kwargs)**

Plot stations.

**Parameters**

- **model\_locations** – Defaults to False.
- **bounding\_box** – Defaults to None.
- **map\_epsg** – Defaults to 4326.
- **\*\*kwargs** – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

*TYPE*

**plot\_strike(\*\*kwargs)**

Plot strike angle

 **See also**

[mtpy.imaging.PlotStrike](#).

**plot\_tipper\_map(\*\*kwargs)**

Plot Phase tensor maps for transfer functions in the working\_dataframe

 **See also**

[mtpy.imaging.PlotPhaseTensorMaps](#).

**Parameters**

**\*\*kwargs** – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

*TYPE*

**remove\_station**(*station\_id*, *survey\_id=None*)

Remove a station from the dictionary based on the key.

**Parameters**

- **station\_id** (*str*) – Station ID.
- **survey\_id** (*str, optional*) – Survey ID, defaults to None.

**rotate**(*rotation\_angle*, *inplace=True*)

Rotate the data by the given angle assuming positive clockwise with north = 0, east = 90.

**Parameters**

- **inplace** – Defaults to True.
- **rotation\_angle** (*TYPE*) – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**property survey\_ids**

Survey IDs for all MT objects.

**Returns**

List of survey IDs.

**Return type**

list

**to\_dataframe**(*utm\_crs=None*, *cols=None*, *impedance\_units='mt'*)

To dataframe.

**Parameters**

- **utm\_crs** (*TYPE, optional*) – DESCRIPTION, defaults to None.
- **cols** (*TYPE, optional*) – DESCRIPTION, defaults to None.
- **impedance\_units** (*str*) – [ ‘mt’ [mV/km/nT] | ‘ohm’ [Ohms] ]

**Returns**

DESCRIPTION.

**Return type**

TYPE

**to\_geo\_df**(*model\_locations=False*, *data\_type='station\_locations'*)

Make a geopandas dataframe for easier GIS manipulation.

**Parameters**

- **model\_locations** (*bool, optional*) – If True returns points in model coordinates,, defaults to False.
- **data\_type** (*string, optional*) – Type of data in GeoDataFrame [ ‘station\_locations’ | ‘phase\_tensor’ | ‘tipper’ | ‘shapefiles’ (both pt and tipper) ], defaults to “station\_locations”.

**Returns**

Geopandas dataframe with requested data in requested coordinates.

**Return type**

geopandas.GeoDataFrame

**to\_modem**(*data\_filename=None*, *\*\*kwargs*)

Create a modem data file.

**Parameters**

- **data\_filename** (*TYPE, optional*) – DESCRIPTION, defaults to None.
- **\*\*kwargs** – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**to\_modem\_data**(*data\_filename=None*, *\*\*kwargs*)

To modem data.

**to\_mt\_dataframe**(*utm\_crs=None*, *impedance\_units='mt'*)

Create an MTDataFrame.

**Parameters**

- **utm\_crs** (*TYPE, optional*) – DESCRIPTION, defaults to None.
- **impedance\_units** (*str*) – [ “mt” [mV/km/nT] | “ohm” [Ohms] ]

**Returns**

DESCRIPTION.

**Return type**

TYPE

**to\_occam2d**(*data\_filename=None*, *\*\*kwargs*)

Write an Occam2D data file.

**Parameters**

- **data\_filename** (*TYPE, optional*) – DESCRIPTION, defaults to None.
- **\*\*kwargs** – DESCRIPTION.

**Returns**

DESCRIPTION.

**Return type**

TYPE

**to\_occam2d\_data**(*data\_filename=None*, *\*\*kwargs*)

To occam2d data.

**to\_shp\_pt\_tipper**(*save\_dir*, *output\_crs=None*, *utm=False*, *pt=True*, *tipper=True*, *periods=None*, *period\_tol=None*, *ellipse\_size=None*, *arrow\_size=None*)

Write phase tensor and tipper shape files.

**Note**

If you have a mixed data set such that the periods do not match up, you should first interpolate onto a common period map and then make shape files. Otherwise you will have a bunch of shapefiles with only a few shapes.

## Parameters

- **arrow\_size** – Defaults to None.
- **ellipse\_size** – Defaults to None.
- **utm** – Defaults to False.
- **save\_dir (string or Path)** – Folder to save shape files to.
- **output\_crs (string, CRS, optional)** – CRS the output shape files will be in, depending of if utm is True or False, defaults to None.
- **pt (bool, optional)** – Make phase tensor shape files, defaults to True.
- **tipper (bool, optional)** – Make tipper shape files, defaults to True.
- **periods (np.ndarray, optional)** – Periods to plot periods within the data, defaults to None.
- **period\_tol (float, optional)** – Tolerance to search around periods, defaults to None.

## Returns

Dictionary of file paths.

## Return type

dictionary

### `to_simpeg_2d(**kwargs)`

Create a data object for Simpeg to work with.

All information is derived from the dataframe. Therefore the user should create the profile, interpolate, estimate model errors from the *MTData* object first before creating the Simpeg2D object.

kwargs include:

- *include\_elevation* -> bool
- *invert\_te* -> bool
- *invert\_tm* -> bool

### `to_simpeg_3d(**kwargs)`

Create a data object that Simpeg can work with.

All information is derived from the dataframe. Therefore the user should interpolate, estimate model errors, etc from the *MTData* object first before creating the Simpeg3D object.

kwargs include:

- *include\_elevation* -> bool
- *geographic\_coordinates* -> bool
- *invert\_z\_xx* -> bool
- *invert\_z\_xy* -> bool
- *invert\_z\_yx* -> bool

- invert\_z\_yy -> bool
- invert\_t\_zx -> bool
- invert\_t\_zy -> bool
- invert\_types = [“real”, “imaginary”]

---

**CHAPTER  
TWO**

---

**INDICES AND TABLES**

- genindex
- modindex
- search



## PYTHON MODULE INDEX

### m

MT, 110  
mtpy, 393  
mtpy.analysis, 84  
mtpy.analysis.residual\_phase\_tensor, 83  
mtpy.core, 145  
mtpy.core.mt, 110  
mtpy.core.mt\_collection, 117  
mtpy.core.mt\_data, 124  
mtpy.core.mt\_dataframe, 137  
mtpy.core.mt\_location, 139  
mtpy.core.mt\_stations, 141  
mtpy.core.transfer\_function, 102  
mtpy.core.transfer\_function.base, 91  
mtpy.core.transfer\_function.pt, 93  
mtpy.core.transfer\_function.tipper, 96  
mtpy.core.transfer\_function.z, 97  
mtpy.core.transfer\_function.z\_analysis, 88  
mtpy.core.transfer\_function.z\_analysis.distortion, 84  
mtpy.core.transfer\_function.z\_analysis.niblett\_bostick, 85  
mtpy.core.transfer\_function.z\_analysis.zinvariants, 87  
mtpy.gis, 162  
mtpy.gis.raster\_tools, 160  
mtpy.gis.shapefile\_creator, 161  
mtpy.imaging, 196  
mtpy.imaging.mtcOLORS, 181  
mtpy.imaging.mtplot\_tools, 173  
mtpy.imaging.mtplot\_tools.arrows, 162  
mtpy.imaging.mtplot\_tools.base, 163  
mtpy.imaging.mtplot\_tools.ellipses, 165  
mtpy.imaging.mtplot\_tools.map\_interpolation\_tools, 166  
mtpy.imaging.mtplot\_tools.plot\_settings, 169  
mtpy.imaging.mtplot\_tools.plotTERS, 170  
mtpy.imaging.mtplot\_tools.utils, 172  
mtpy.imaging.plot\_mt\_response, 181  
mtpy.imaging.plot\_mt\_responses, 182  
mtpy.imaging.plot\_penetration\_depth\_1d, 183  
mtpy.imaging.plot\_penetration\_depth\_map, 183  
mtpy.imaging.plot\_phase\_tensor\_maps, 184  
mtpy.imaging.plot\_phase\_tensor\_pseudosection, 185  
mtpy.imaging.plot\_pseudosection, 186  
mtpy.imaging.plot\_pt, 186  
mtpy.imaging.plot\_residual\_pt\_maps, 186  
mtpy.imaging.plot\_residual\_pt\_ps, 188  
mtpy.imaging.plot\_rephase\_maps, 191  
mtpy.imaging.plot\_spectrogram, 192  
mtpy.imaging.plot\_stations, 193  
mtpy.imaging.plot\_strike, 193  
mtpy.modeling, 327  
mtpy.modeling.errors, 303  
mtpy.modeling.gocad, 305  
mtpy.modeling.mesh\_tools, 305  
mtpy.modeling.modem, 223  
mtpy.modeling.modem.config, 206  
mtpy.modeling.modem.control\_fwd, 206  
mtpy.modeling.modem.control\_inv, 207  
mtpy.modeling.modem.convariance, 208  
mtpy.modeling.modem.data, 209  
mtpy.modeling.modem.exception, 210  
mtpy.modeling.modem.model, 211  
mtpy.modeling.modem.residual, 219  
mtpy.modeling.modem.station, 221  
mtpy.modeling.occam1d, 249  
mtpy.modeling.occam1d.data, 237  
mtpy.modeling.occam1d.model, 240  
mtpy.modeling.occam1d.plot\_12, 243  
mtpy.modeling.occam1d.plot\_response, 244  
mtpy.modeling.occam1d.run, 247  
mtpy.modeling.occam1d.startup, 247  
mtpy.modeling.occam2d, 272  
mtpy.modeling.occam2d.data, 260  
mtpy.modeling.occam2d.mesh, 262  
mtpy.modeling.occam2d.model, 266  
mtpy.modeling.occam2d.regularization, 268  
mtpy.modeling.occam2d.startup, 270  
mtpy.modeling.plots, 284  
mtpy.modeling.plots.plot\_mesh, 283  
mtpy.modeling.plots.plot\_modem\_rms, 283  
mtpy.modeling.simpeg, 292

`mtpy.modeling.simpeg.make_2d_mesh`, 290  
`mtpy.modeling.simpeg.recipes`, 286  
`mtpy.modeling.simpeg.recipes.inversion_1d`,  
    285  
`mtpy.modeling.structured_mesh_3d`, 307  
`mtpy.modeling.winglink`, 319  
`mtpy.modeling.winglinktools`, 326  
`mtpy.modeling.ws3dinv`, 297  
`mtpy.modeling.ws3dinv.data`, 292  
`mtpy.modeling.ws3dinv.startup`, 295  
`mtpy.modeling.ws3dinv.stations`, 296  
`mtpy.mtpy_globals`, 393  
`mtpy.processing`, 370  
`mtpy.processing.aurora`, 341  
`mtpy.processing.aurora.process_aurora`, 339  
`mtpy.processing.base`, 341  
`mtpy.processing.birrp`, 341  
`mtpy.processing.filter`, 347  
`mtpy.processing.kernel_dataset`, 350  
`mtpy.processing.run_summary`, 356  
`mtpy.processing.tf`, 358  
`mtpy.utils`, 393  
`mtpy.utils.basemap_tools`, 377  
`mtpy.utils.calculator`, 378  
`mtpy.utils.concatenate_input`, 380  
`mtpy.utils.configfile`, 381  
`mtpy.utils.edi_folders`, 383  
`mtpy.utils.estimate_tf_quality_factor`, 383  
`mtpy.utils.exceptions`, 385  
`mtpy.utils.filehandling`, 386  
`mtpy.utils.gis_tools`, 389  
`mtpy.utils.matplotlib_utils`, 391  
`mtpy.utils.mtpy_decorator`, 391  
`mtpy.utils.plot_rms_iterations`, 391  
`mtpy.utils.sensor_orientation_correction`, 392

**t**

`TFBase`, 91

# INDEX

## A

active\_cell\_index (*mtpy.modeling.simpeg.make\_2d\_mesh*.QuadTreeMesh property), 290  
active\_cell\_index (*mtpy.modeling.simpeg.make\_2d\_mesh*.property), 291  
active\_map (*mtpy.modeling.simpeg.recipes.Simpeg2D*.property), 287  
adaptive\_notch\_filter() (in *mtpy.processing.filter*), 347  
add\_basemap\_frame() (in *mtpy.utils.basemap\_tools*), 377  
add\_colorbar\_axis() (in *mtpy.imaging.mplot\_tools.utils*), 172  
add\_columns\_for\_processing() (*mtpy.processing.kernel\_dataset.KernelDataset*.method), 351  
add\_columns\_for\_processing() (*mtpy.processing.KernelDataset*.method), 372  
add\_dict() (*mtpy.modeling.modem.config.ModEMConfig*.method), 206  
add\_dict() (*mtpy.modeling.modem.ModEMConfig*.method), 226  
add\_elevation() (*mtpy.modeling.occam2d.Mesh*.method), 274  
add\_elevation() (*mtpy.modeling.occam2d.mesh.Mesh*.method), 264  
add\_layers\_to\_mesh() (*mtpy.modeling.modem.Model*.method), 229  
add\_layers\_to\_mesh() (*mtpy.modeling.modem.model.Model*.method), 213  
add\_layers\_to\_mesh() (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D*.method), 309  
add\_layers\_to\_mesh() (*mtpy.modeling.StructuredGrid3D*.method), 329  
add\_model\_error() (*mtpy.core.mt.MT*.method), 111  
add\_model\_error() (*mtpy.MT*.method), 394  
add\_raster() (in module *mtpy.imaging.mplot\_tools*), 177  
add\_raster() (in *mtpy.imaging.mplot\_tools.plotters*), 170  
add\_raster() (*mtpy.imaging.mplot\_tools.base.PlotBaseMaps*.method), 164  
add\_raster() (*mtpy.imaging.mplot\_tools.PlotBaseMaps*.method), 176  
add\_station() (*mtpy.core.mt\_data.MTData*.method), 124  
add\_station() (*mtpy.MTData*.method), 407  
add\_tf() (*mtpy.core.mt\_collection.MTCollection*.method), 118  
add\_tf() (*mtpy.core.mt\_data.MTData*.method), 124  
add\_tf() (*mtpy.MTCollection*.method), 400  
add\_tf() (*mtpy.MTData*.method), 407  
add\_topography\_from\_data() (*mtpy.modeling.modem.Model*.method), 229  
add\_topography\_from\_data() (*mtpy.modeling.modem.model.Model*.method), 213  
add\_topography\_from\_data() (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D*.method), 310  
add\_topography\_from\_data() (*mtpy.modeling.StructuredGrid3D*.method), 330  
add\_topography\_to\_model() (*mtpy.modeling.modem.Model*.method), 230  
add\_topography\_to\_model() (*mtpy.modeling.modem.model.Model*.method), 214  
add\_topography\_to\_model() (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D*.method), 310  
add\_topography\_to\_model() (*mtpy.modeling.StructuredGrid3D*.method), 330  
add\_white\_noise() (*mtpy.core.mt.MT*.method), 112  
add\_white\_noise() (*mtpy.core.mt\_data.MTData*.method), 125  
add\_white\_noise() (*mtpy.MT*.method), 394

add\_white\_noise() (*mtpy.MTData* method), 407  
alpha (*mtpy.core.PhaseTensor* property), 152  
alpha (*mtpy.core.transfer\_function.PhaseTensor* property), 102  
alpha (*mtpy.core.transfer\_function.pt.PhaseTensor* property), 94  
alpha\_error (*mtpy.core.PhaseTensor* property), 152  
alpha\_error (*mtpy.core.transfer\_function.PhaseTensor* property), 102  
alpha\_error (*mtpy.core.transfer\_function.pt.PhaseTensor* property), 94  
alpha\_model\_error (*mtpy.core.PhaseTensor* property), 152  
alpha\_model\_error (*mtpy.core.transfer\_function.PhaseTensor* property), 102  
alpha\_model\_error (*mtpy.core.transfer\_function.pt.PhaseTensor* property), 94  
amplitude (*mtpy.core.Tipper* property), 154  
amplitude (*mtpy.core.transfer\_function.Tipper* property), 105  
amplitude (*mtpy.core.transfer\_function.tipper.Tipper* property), 96  
amplitude\_error (*mtpy.core.Tipper* property), 155  
amplitude\_error (*mtpy.core.transfer\_function.Tipper* property), 105  
amplitude\_error (*mtpy.core.transfer\_function.tipper.Tipper* property), 96  
amplitude\_model\_error (*mtpy.core.Tipper* property), 155  
amplitude\_model\_error (*mtpy.core.transfer\_function.Tipper* property), 105  
amplitude\_model\_error (*mtpy.core.transfer\_function.tipper.Tipper* property), 96  
angle\_error (*mtpy.core.Tipper* property), 155  
angle\_error (*mtpy.core.transfer\_function.Tipper* property), 105  
angle\_error (*mtpy.core.transfer\_function.tipper.Tipper* property), 96  
angle\_imag (*mtpy.core.Tipper* property), 155  
angle\_imag (*mtpy.core.transfer\_function.Tipper* property), 105  
angle\_imag (*mtpy.core.transfer\_function.tipper.Tipper* property), 96  
angle\_model\_error (*mtpy.core.Tipper* property), 155  
angle\_model\_error (*mtpy.core.transfer\_function.Tipper* property), 105  
angle\_model\_error (*mtpy.core.transfer\_function.tipper.Tipper* property), 96  
angle\_real (*mtpy.core.Tipper* property), 155  
angle\_real (*mtpy.core.transfer\_function.Tipper* property), 105  
angle\_real (*mtpy.core.transfer\_function.tipper.Tipper* property), 96  
property), 96  
anisotropic\_imag (*mtpy.core.transfer\_function.z\_analysis.ZInvariants* property), 88  
anisotropic\_imag (*mtpy.core.transfer\_function.z\_analysis.zinvariants.ZI* property), 87  
anisotropic\_real (*mtpy.core.transfer\_function.z\_analysis.ZInvariants* property), 89  
anisotropic\_real (*mtpy.core.transfer\_function.z\_analysis.zinvariants.ZI* property), 88  
apply\_bbox() (*mtpy.core.mt\_collection.MTCollection* method), 118  
apply\_bbox() (*mtpy.MTCollection* method), 400  
apply\_bounding\_box() (*mtpy.core.mt\_data.MTData* method), 125  
apply\_bounding\_box() (*mtpy.MTData* method), 407  
array2raster() (*in module mtpy.gis.raster\_tools*), 160  
arrow\_imag\_properties  
    (*mtpy.imaging.mplot\_tools.plot\_settings.PlotSettings* property), 169  
arrow\_imag\_properties  
    (*mtpy.imaging.mplot\_tools.PlotSettings* property), 176  
arrow\_real\_properties  
    (*mtpy.imaging.mplot\_tools.plot\_settings.PlotSettings* property), 169  
arrow\_real\_properties  
    (*mtpy.imaging.mplot\_tools.PlotSettings* property), 177  
assert\_elevation\_value() (*in module mtpy.utils.gis\_tools*), 389  
assert\_lat\_value() (*in module mtpy.utils.gis\_tools*), 389  
assert\_lon\_value() (*in module mtpy.utils.gis\_tools*), 389  
assert\_minutes() (*in module mtpy.utils.gis\_tools*), 389  
assert\_seconds() (*in module mtpy.utils.gis\_tools*), 389  
assign\_resistivity\_from\_surface\_data()  
    (*mtpy.modeling.modem.Model* method), 230  
assign\_resistivity\_from\_surface\_data()  
    (*mtpy.modeling.modem.model.Model* method), 214  
assign\_resistivity\_from\_surface\_data()  
    (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D* method), 311  
assign\_resistivity\_from\_surface\_data()  
    (*mtpy.modeling.StructuredGrid3D* method), 331  
AuroraProcessing (*class in mtpy.processing*), 370  
AuroraProcessing  
    (*class in mtpy.processing.aurora.process\_aurora*), 339  
average\_stations() (*mtpy.core.mt\_collection.MTCollection* method), 118

**A**

- average\_stations() (*mtpy.MTCollection method*), build\_regularization()
  - 401
- azimuth (*mtpy.core.PhaseTensor property*), 152
- azimuth (*mtpy.core.transfer\_function.PhaseTensor property*), 102
- azimuth (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 94
- azimuth\_error (*mtpy.core.PhaseTensor property*), 153
- azimuth\_error (*mtpy.core.transfer\_function.PhaseTensor property*), 103
- azimuth\_error (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 94
- azimuth\_model\_error (*mtpy.core.PhaseTensor property*), 153
- azimuth\_model\_error (*mtpy.core.transfer\_function.PhaseTensor property*), 103
- azimuth\_model\_error (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 94

**B**

- BaseProcessing (*class in mtpy.processing.base*), 341
- beta (*mtpy.core.PhaseTensor property*), 153
- beta (*mtpy.core.transfer\_function.PhaseTensor property*), 103
- beta (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 94
- beta\_error (*mtpy.core.PhaseTensor property*), 153
- beta\_error (*mtpy.core.transfer\_function.PhaseTensor property*), 103
- beta\_error (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 94
- beta\_model\_error (*mtpy.core.PhaseTensor property*), 153
- beta\_model\_error (*mtpy.core.transfer\_function.PhaseTensor property*), 103
- beta\_model\_error (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 94
- beta\_schedule (*mtpy.modeling.simpeg.recipes.Simpeg2D property*), 287
- BIRRPPParameterError, 341
- BIRRPPParameters (*class in mtpy.processing.birrp*), 341
- build\_mesh() (*mtpy.modeling.occam2d.Mesh method*), 274
- build\_mesh() (*mtpy.modeling.occam2d.mesh.Mesh method*), 264
- build\_model() (*mtpy.modeling.occam2d.model.Occam2DModel method*), 267
- build\_model() (*mtpy.modeling.occam2d.Occam2DModel method*), 279
- build\_regularization() (*mtpy.modeling.occam2d.Regularization method*), 281

**C**

- calculate\_depth\_of\_investigation() (*in module mtpy.core.transfer\_function.z\_analysis*), 89
- calculate\_depth\_of\_investigation() (*in module mtpy.core.transfer\_function.z\_analysis.niblettbostick*), 85
- calculate\_depth\_sensitivity() (*in module mtpy.core.transfer\_function.z\_analysis.niblettbostick*), 86
- calculate\_niblett\_bostick\_depth() (*in module mtpy.core.transfer\_function.z\_analysis.niblettbostick*), 86
- calculate\_niblett\_bostick\_resistivity\_derivatives() (*in module mtpy.core.transfer\_function.z\_analysis.niblettbostick*), 87
- calculate\_niblett\_bostick\_resistivity\_weidelt() (*in module mtpy.core.transfer\_function.z\_analysis.niblettbostick*), 87
- calculate\_rel\_locations() (*mtpy.modeling.modem.station.Stations method*), 221
- calculate\_rms() (*mtpy.modeling.modem.Residual method*), 236
- calculate\_rms() (*mtpy.modeling.modem.residual.Residual method*), 220
- center\_point (*mtpy.core.mt\_stations.MTStations property*), 141
- center\_point (*mtpy.core.MTStations property*), 148
- center\_point (*mtpy.modeling.modem.station.Stations property*), 221
- center\_stations() (*mtpy.core.mt\_stations.MTStations method*), 141
- center\_stations() (*mtpy.core.MTStations method*), 148
- centre\_point() (*in module mtpy.utils.calculator*), 378
- check\_for\_duplicates() (*mtpy.core.mt\_collection.MTCollection method*), 118
- check\_for\_duplicates() (*mtpy.MTCollection method*), 401
- clone() (*mtpy.processing.kernel\_dataset.KernelDataset method*), 351
- clone() (*mtpy.processing.KernelDataset method*), 372
- clone() (*mtpy.processing.run\_summary.RunSummary method*), 356
- clone() (*mtpy.processing.RunSummary method*), 376

clone\_dataframe() (`mtpy.processing.kernel_dataset.KernelDataset` method), 146  
    method), 351  
clone\_dataframe() (`mtpy.processing.KernelDataset` method), 372  
clone\_empty() (`mtpy.core.mt.MT` method), 112  
clone\_empty() (`mtpy.core.mt_data.MTData` method), 125  
clone\_empty() (`mtpy.MT` method), 394  
clone\_empty() (`mtpy.MTData` method), 408  
close\_collection() (`mtpy.core.mt_collection.MTCollection` method), 119  
close\_collection() (`mtpy.MTCollection` method), 401  
close\_mth5s() (`mtpy.processing.kernel_dataset.KernelDataset` method), 351  
close\_mth5s() (`mtpy.processing.KernelDataset` method), 372  
cmap\_discretize() (in module `mtpy.imaging.mtcolors`), 181  
comp\_list (`mtpy.processing.birrp.ScriptFile` property), 346  
comps (`mtpy.core.transfer_function.base.TFBase` property), 91  
compute\_absolute\_error() (`mtpy.modeling.errors.ModelErrors` method), 303  
compute\_arithmetic\_mean\_error() (`mtpy.modeling.errors.ModelErrors` method), 303  
compute\_determinant\_error() (in module `mtpy.utils.calculator`), 378  
compute\_eigen\_value\_error() (`mtpy.modeling.errors.ModelErrors` method), 303  
compute\_error() (`mtpy.modeling.errors.ModelErrors` method), 303  
compute\_geometric\_mean\_error() (`mtpy.modeling.errors.ModelErrors` method), 304  
compute\_lonlat0\_from\_modem\_data() (in module `mtpy.utils.basemap_tools`), 377  
compute\_map\_extent\_from\_modem\_data() (in module `mtpy.utils.basemap_tools`), 377  
compute\_median\_error() (`mtpy.modeling.errors.ModelErrors` method), 304  
compute\_model\_errors() (`mtpy.core.mt_data.MTData` method), 125  
compute\_model\_errors() (`mtpy.MTData` method), 408  
compute\_model\_location() (`mtpy.core.mt_location.MTLocation` method), 139  
compute\_model\_location() (`mtpy.core.MTLocation` method), 146  
    method), 112  
compute\_model\_t\_errors() (`mtpy.core.mt.MT` method), 394  
compute\_model\_z\_errors() (`mtpy.core.mt.MT` method), 112  
compute\_model\_z\_errors() (`mtpy.MT` method), 395  
compute\_percent\_error() (`mtpy.modeling.errors.ModelErrors` method), 304  
compute\_relative\_locations() (`mtpy.core.mt_stations.MTStations` method), 141  
compute\_relative\_locations() (`mtpy.core.MTStations` method), 148  
compute\_residual\_pt() (`mtpy.analysis.residual_phase_tensor.ResidualPhaseTensor` method), 83  
compute\_row\_error() (`mtpy.modeling.errors.ModelErrors` method), 304  
compute\_statistics() (`mtpy.utils.estimate_tf_quality_factor.EMTFStats` method), 383  
compute\_tick\_interval\_from\_map\_extent() (in module `mtpy.utils.basemap_tools`), 377  
concatenate\_log\_files() (in module `mtpy.utils.plot_rms_iterations`), 391  
conductivity\_map (`mtpy.modeling.simpeg.recipes.Simpeg2D` property), 288  
control\_fn (`mtpy.modeling.modem.control_fwd.ControlFwd` property), 206  
control\_fn (`mtpy.modeling.modem.control_inv.ControlInv` property), 207  
control\_fn (`mtpy.modeling.modem.ControlFwd` property), 223  
control\_fn (`mtpy.modeling.modem.ControlInv` property), 224  
ControlFwd (class in `mtpy.modeling.modem`), 223  
ControlFwd (class in `mtpy.modeling.modem.control_fwd`), 206  
ControlInv (class in `mtpy.modeling.modem`), 223  
ControlInv (class in `mtpy.modeling.modem.control_inv`), 207  
convert\_model\_to\_int() (`mtpy.modeling.structured_mesh_3d.StructuredGrid3D` method), 311  
convert\_model\_to\_int() (`mtpy.modeling.StructuredGrid3D` method), 331  
convert\_position\_float2str() (in module `mtpy.utils.gis_tools`), 390  
convert\_position\_str2float() (in module `mtpy.utils.gis_tools`), 390

coordinate\_reference\_frame (mtpy.core.mt.MT property), 112

coordinate\_reference\_frame (mtpy.core.mt\_data.MTData property), 126

coordinate\_reference\_frame (mtpy.MT property), 395

coordinate\_reference\_frame (mtpy.MTData property), 408

copy() (mtpy.core.mt.MT method), 112

copy() (mtpy.core.mt\_data.MTData method), 126

copy() (mtpy.core.mt\_location.MTLocation method), 139

copy() (mtpy.core.mt\_stations.MTStations method), 141

copy() (mtpy.core.MTLocation method), 147

copy() (mtpy.core.MTStations method), 148

copy() (mtpy.core.transfer\_function.base.TFBase method), 91

copy() (mtpy.MT method), 395

copy() (mtpy.MTData method), 409

correct4sensor\_orientation() (in module mtpy.utils.sensor\_orientation\_correction), 392

cov\_fn (mtpy.modeling.modem.convariance.Covariance property), 208

cov\_fn (mtpy.modeling.modem.Covariance property), 225

Covariance (class in mtpy.modeling.modem), 224

Covariance (class in mtpy.modeling.modem.convariance), 208

create\_config() (mtpy.processing.aurora.process\_aurora.AuroraProcessing method), 339

create\_config() (mtpy.processing.AuroraProcessing method), 370

create\_kernel\_dataset() (mtpy.processing.aurora.process\_aurora.AuroraProcessing method), 339

create\_kernel\_dataset() (mtpy.processing.AuroraProcessing method), 370

cull\_from\_difference() (mtpy.modeling.simpeg.recipes.inversion\_Id.SimpegID method), 285

cull\_from\_difference() (mtpy.modeling.simpeg.recipes.SimpegID method), 286

cull\_from\_interpolated() (mtpy.modeling.simpeg.recipes.inversion\_Id.SimpegID method), 285

cull\_from\_interpolated() (mtpy.modeling.simpeg.recipes.SimpegID method), 286

cull\_from\_model() (mtpy.modeling.simpeg.recipes.inversion\_Id.SimpegID method), 285

cull\_from\_model() (mtpy.modeling.simpeg.recipes.SimpegID method), 139

method), 286

**D**

Data (class in mtpy.modeling.modem), 225

Data (class in mtpy.modeling.modem.data), 209

Data (class in mtpy.utils.concatenate\_input), 381

data (mtpy.modeling.errors.ModelErrors property), 304

data (mtpy.modeling.simpeg.recipes.inversion\_Id.SimpegID property), 285

data (mtpy.modeling.simpeg.recipes.SimpegID property), 286

data\_error (mtpy.modeling.simpeg.recipes.inversion\_Id.SimpegID property), 285

data\_error (mtpy.modeling.simpeg.recipes.SimpegID property), 286

data\_filename (mtpy.modeling.modem.Data property), 225

data\_filename (mtpy.modeling.modem.data.Data property), 209

data\_filename (mtpy.modeling.occam2d.data.Occam2DData property), 261

data\_filename (mtpy.modeling.occam2d.Occam2DData property), 277

data\_filename (mtpy.modeling.ws3dinv.data.WSData property), 294

data\_filename (mtpy.modeling.ws3dinv.WSData property), 300

data\_fn (mtpy.modeling.occam1d.Occam1DStartup property), 255

data\_fn (mtpy.modeling.occam1d.startup.Occam1DStartup property), 247

data\_misfit (mtpy.modeling.simpeg.recipes.Simpeg2D property), 288

DataError, 210, 226

dataframe (mtpy.core.mt\_collection.MTCollection property), 119

dataframe (mtpy.modeling.modem.Data property), 225

dataframe (mtpy.modeling.modem.data.Data property), 209

dataframe (mtpy.modeling.occam2d.data.Occam2DData property), 261

dataframe (mtpy.modeling.occam2d.Occam2DDData property), 277

dataframe (mtpy.modeling.plots.plot\_modem\_rms.PlotRMS property), 283

dataframe (mtpy.modeling.plots.PlotRMS property), 284

dataframe (mtpy.modeling.ws3dinv.data.WSData property), 294

dataframe (mtpy.modeling.ws3dinv.WSData property), 300

dataframe (mtpy.core.mt.MTCollection property), 401

datum\_crs (mtpy.core.mt\_location.MTLocation property), 139

datum\_crs (*mtpy.core.mt\_stations.MTStations* property), 141  
datum\_crs (*mtpy.core.MTLocation* property), 147  
datum\_crs (*mtpy.core.MTStations* property), 149  
datum\_epsg (*mtpy.core.mt\_dataframe.MTDataFrame* property), 137  
datum\_epsg (*mtpy.core.mt\_location.MTLocation* property), 139  
datum\_epsg (*mtpy.core.mt\_stations.MTStations* property), 142  
datum\_epsg (*mtpy.core.MTDataFrame* property), 145  
datum\_epsg (*mtpy.core.MTLocation* property), 147  
datum\_epsg (*mtpy.core.MTStations* property), 149  
datum\_name (*mtpy.core.mt\_location.MTLocation* property), 139  
datum\_name (*mtpy.core.mt\_stations.MTStations* property), 142  
datum\_name (*mtpy.core.MTLocation* property), 147  
datum\_name (*mtpy.core.MTStations* property), 149  
`dctrend()` (*in module mtpy.processing.tf*), 358  
`decimate()` (*in module mtpy.processing.tf*), 358  
`deltat` (*mtpy.processing.birrp.ScriptFile* property), 346  
deprecated (class in *mtpy.utils.mtpy\_decorator*), 391  
`depth_units` (*mtpy.imaging.plot\_penetration\_depth\_1d.PlotPenetrationDepth1D* property), 183  
`depth_units` (*mtpy.imaging.plot\_penetration\_depth\_map.PlotPenetrationDepthMap* property), 184  
`depth_units` (*mtpy.imaging.PlotPenetrationDepth1D* property), 197  
`depth_units` (*mtpy.imaging.PlotPenetrationDepthMap* property), 197  
`det` (*mtpy.core.PhaseTensor* property), 153  
`det` (*mtpy.core.transfer\_function.PhaseTensor* property), 103  
`det` (*mtpy.core.transfer\_function.pt.PhaseTensor* property), 94  
`det` (*mtpy.core.transfer\_function.Z* property), 106  
`det` (*mtpy.core.transfer\_function.z.Z* property), 98  
`det` (*mtpy.core.Z* property), 156  
`det_error` (*mtpy.core.PhaseTensor* property), 153  
`det_error` (*mtpy.core.transfer\_function.PhaseTensor* property), 103  
`det_error` (*mtpy.core.transfer\_function.pt.PhaseTensor* property), 94  
`det_error` (*mtpy.core.transfer\_function.Z* property), 106  
`det_error` (*mtpy.core.transfer\_function.z.Z* property), 98  
`det_error` (*mtpy.core.Z* property), 156  
`det_error_bar_properties` (*mtpy.imaging.mtplot\_tools.plot\_settings.PlotSettings* property), 169  
`det_error_bar_properties` (*mtpy.imaging.mtplot\_tools.PlotSettings* property), 177  
`det_model_error` (*mtpy.core.PhaseTensor* property), 153  
`det_model_error` (*mtpy.core.transfer\_function.PhaseTensor* property), 103  
`det_model_error` (*mtpy.core.transfer\_function.pt.PhaseTensor* property), 94  
`det_model_error` (*mtpy.core.transfer\_function.Z* property), 106  
`det_model_error` (*mtpy.core.transfer\_function.z.Z* property), 98  
`det_model_error` (*mtpy.core.Z* property), 156  
`df` (*mtpy.processing.kernel\_dataset.KernelDataset* property), 351  
`df` (*mtpy.processing.KernelDataset* property), 372  
`df` (*mtpy.processing.run\_summary.RunSummary* property), 357  
`df` (*mtpy.processing.RunSummary* property), 376  
`dimensionality` (*mtpy.core.transfer\_function.z.z\_analysis.ZInvariants* property), 89  
`dimensionality` (*mtpy.core.transfer\_function.z.z\_analysis.zinvariants.ZInvariants* property), 88  
`directives` (*mtpy.modeling.simpeg.recipes.Simpeg2D* property), 288  
~~`drop_no_data_rows()`~~ (*mtpy.processing.run\_summary.RunSummary* method), 373  
`drop_no_data_rows()` (*mtpy.processing.RunSummary* method), 376  
`drop_runs_shorter_than()` (*mtpy.processing.kernel\_dataset.KernelDataset* method), 351  
`drop_runs_shorter_than()` (*mtpy.processing.KernelDataset* method), 372  
`dwindow()` (*in module mtpy.processing.tf*), 359  
`dx` (*mtpy.modeling.simpeg.make\_2d\_mesh.QuadTreeMesh* property), 290  
`dx` (*mtpy.modeling.simpeg.make\_2d\_mesh.StructuredMesh* property), 291  
`dz` (*mtpy.modeling.simpeg.make\_2d\_mesh.QuadTreeMesh* property), 290

## E

`east` (*mtpy.core.mt\_dataframe.MTDataFrame* property), 137  
`east` (*mtpy.core.mt\_location.MTLocation* property), 139  
`east` (*mtpy.core.MTDataFrame* property), 145  
`east` (*mtpy.core.MTLocation* property), 147  
`east` (*mtpy.modeling.modem.station.Stations* property), 221  
`eccentricity` (*mtpy.core.PhaseTensor* property), 153  
`eccentricity` (*mtpy.core.transfer\_function.PhaseTensor* property), 103

**eccentricity** (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 94  
**eccentricity\_error** (*mtpy.core.PhaseTensor property*), 153  
**eccentricity\_error** (*mtpy.core.transfer\_function.PhaseTensor property*), 103  
**eccentricity\_error** (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 94  
**eccentricity\_error** (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 94  
**eccentricity\_model\_error** (*mtpy.core.PhaseTensor property*), 153  
**eccentricity\_model\_error** (*mtpy.core.transfer\_function.PhaseTensor property*), 103  
**eccentricity\_model\_error** (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 94  
**EdiFolders** (*class in mtpy.utils.edi\_folders*), 383  
**edit\_curve()** (*mtpy.core.mt.MT method*), 112  
**edit\_curve()** (*mtpy.MT method*), 395  
**EDL\_get\_starttime\_fromfilename()** (*in module mtpy.utils.filehandling*), 386  
**EDL\_get\_stationname\_fromfilename()** (*in module mtpy.utils.filehandling*), 386  
**EDL\_make\_dayfiles()** (*in module mtpy.utils.filehandling*), 386  
**EDL\_make\_Nhour\_files()** (*in module mtpy.utils.filehandling*), 386  
**electric\_twist** (*mtpy.core.transfer\_function.z\_analysis.ZInvariant property*), 89  
**electric\_twist** (*mtpy.core.transfer\_function.z\_analysis.zinvariants.ZInvariants property*), 88  
**elev** (*mtpy.modeling.modem.station.Stations property*), 221  
**elevation** (*mtpy.core.mt\_dataframe.MTDataFrame property*), 137  
**elevation** (*mtpy.core.mt\_location.MTLocation property*), 140  
**elevation** (*mtpy.core.MTDataFrame property*), 145  
**elevation** (*mtpy.core.MTLocation property*), 147  
**ellipse\_cmap\_bounds** (*mtpy.imaging.mtplot\_tools.ellipses.MTEllipse property*), 166  
**ellipse\_cmap\_bounds** (*mtpy.imaging.mtplot\_tools.MTEllipse property*), 174  
**ellipse\_cmap\_n\_segments** (*mtpy.imaging.mtplot\_tools.ellipses.MTEllipse property*), 166  
**ellipse\_cmap\_n\_segments** (*mtpy.imaging.mtplot\_tools.MTEllipse property*), 174  
**ellipse\_properties** (*mtpy.imaging.mtplot\_tools.ellipses.MTEllipse property*), 166  
**ellipse\_properties** (*mtpy.imaging.mtplot\_tools.MTEllipse property*), 166  
**ellipticity** (*mtpy.core.PhaseTensor property*), 153  
**ellipticity** (*mtpy.core.transfer\_function.PhaseTensor property*), 103  
**ellipticity** (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 94  
**ellipticity\_error** (*mtpy.core.PhaseTensor property*), 153  
**ellipticity\_error** (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 103  
**ellipticity\_error** (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 94  
**ellipticity\_model\_error** (*mtpy.core.PhaseTensor property*), 153  
**ellipticity\_model\_error** (*mtpy.core.transfer\_function.PhaseTensor property*), 103  
**ellipticity\_model\_error** (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 94  
**ellipticity\_model\_error** (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 153  
**ellipticity\_model\_error** (*mtpy.core.transfer\_function.PhaseTensor property*), 103  
**ellipticity\_model\_error** (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 94  
**EMTFStats** (*class in mtpy.utils.estimate\_tf\_quality\_factor*), 383  
**error\_parameters** (*mtpy.modeling.errors.ModelErrors property*), 304  
**error\_type** (*mtpy.modeling.errors.ModelErrors property*), 304  
**error\_value** (*mtpy.modeling.errors.ModelErrors property*), 304  
**estimate\_arrow\_size()**  
**estimate\_shapefile\_creator()** (*ZipyagisShapefileCreator method*), 161  
**estimate\_data\_quality()**  
**estimate\_depth\_of\_investigation()** (*mtpy.core.transfer\_function.Z method*), 106  
**estimate\_depth\_of\_investigation()** (*mtpy.core.transfer\_function.z.Z method*), 98  
**estimate\_depth\_of\_investigation()** (*mtpy.core.Z method*), 156  
**estimate\_dimensionality()** (*mtpy.core.transfer\_function.Z method*), 106  
**estimate\_dimensionality()** (*mtpy.core.transfer\_function.z.Z method*), 98  
**estimate\_dimensionality()** (*mtpy.core.Z method*), 156  
**estimate\_distortion()**  
**estimate\_distortion()** (*mtpy.core.transfer\_function.Z method*), 106

(*mtpy.core.transfer\_function.z.Z* method), 98  
estimate\_distortion() (*mtpy.core.Z* method), 156  
estimate\_ellipse\_size()  
    (*mtpy.gis.shapefile\_creator.ShapefileCreator* method), 161  
estimate\_quality\_factor()  
    (*mtpy.utils.estimate\_tf\_quality\_factor.EMTFStats* method), 384  
estimate\_skin\_depth()  
    (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D* method), 311  
estimate\_skin\_depth()  
    (*mtpy.modeling.StructuredGrid3D* method), 331  
estimate\_spatial\_static\_shift()  
    (*mtpy.core.mt\_data.MTData* method), 126  
estimate\_spatial\_static\_shift() (*mtpy.MTData* method), 409  
estimate\_starting\_rho()  
    (*mtpy.core.mt\_data.MTData* method), 127  
estimate\_starting\_rho() (*mtpy.MTData* method), 409  
estimate\_tf\_quality() (*mtpy.core.mt.MT* method), 112  
estimate\_tf\_quality() (*mtpy.MT* method), 395  
ex\_metadata (*mtpy.core.mt.MT* property), 113  
ex\_metadata (*mtpy.MT* property), 395  
exponent\_map (*mtpy.modeling.simpeg.recipes.Simpeg2D* property), 288  
extract\_run\_summaries\_from\_mth5s() (in module  
    *mtpy.processing.run\_summary*), 357  
extract\_run\_summary\_from\_mth5() (in module  
    *mtpy.processing.run\_summary*), 358  
ey\_metadata (*mtpy.core.mt.MT* property), 113  
ey\_metadata (*mtpy.MT* property), 395

**F**

find\_distortion() (in module  
    *mtpy.core.transfer\_function.z\_analysis*), 90  
find\_distortion() (in module  
    *mtpy.core.transfer\_function.z\_analysis.distortion*), 84  
find\_edi\_folders() (*mtpy.utils.edi\_folders.EdiFolders* method), 383  
find\_flipped\_phase() (*mtpy.core.mt.MT* method), 113  
find\_flipped\_phase() (*mtpy.MT* method), 395  
fix\_data\_file() (*mtpy.modeling.modem.Data* method), 226  
fix\_data\_file() (*mtpy.modeling.modem.data.Data* method), 209  
FixPointNormalize (class in *mtpy.imaging.mtcolors*), 181  
flip\_phase() (*mtpy.core.mt.MT* method), 113  
flip\_phase() (*mtpy.MT* method), 395  
floor (*mtpy.modeling.errors.ModelErrors* property), 304  
font\_dict (*mtpy.imaging.mtplot\_tools.plot\_settings.PlotSettings* property), 169  
font\_dict (*mtpy.imaging.mtplot\_tools.PlotSettings* property), 177  
for\_shapefiles (*mtpy.core.mt\_dataframe.MTDataFrame* property), 138  
foldShapefiles (*mtpy.core.MTDataFrame* property), 145  
frequencies (*mtpy.modeling.occam2d.data.Occam2DData* property), 261  
frequencies (*mtpy.modeling.occam2d.Occam2DData* property), 277  
frequencies (*mtpy.modeling.simpeg.recipes.inversion\_1d.Simpeg1D* property), 285  
frequencies (*mtpy.modeling.simpeg.recipes.Simpeg1D* property), 286  
frequency (*mtpy.core.mt\_dataframe.MTDataFrame* property), 138  
frequency (*mtpy.core.MTDataFrame* property), 145  
frequency (*mtpy.core.transfer\_function.base.TFBase* property), 91  
frequency\_max (*mtpy.modeling.simpeg.make\_2d\_mesh.StructuredMesh* property), 291  
frequency\_min (*mtpy.modeling.simpeg.make\_2d\_mesh.StructuredMesh* property), 291  
from\_dataframe() (*mtpy.core.mt.MT* method), 114  
from\_dataframe() (*mtpy.core.mt\_data.MTData* method), 127  
from\_dataframe() (*mtpy.core.transfer\_function.base.TFBase* method), 91  
from\_dataframe() (*mtpy.MT* method), 396  
from\_dataframe() (*mtpy.MTData* method), 409  
from\_dict() (*mtpy.processing.birrp.BIRPPParameters* method), 341  
from\_gocad\_sgrid() (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D* method), 311  
from\_gocad\_sgrid() (*mtpy.modeling.StructuredGrid3D* method), 331  
from\_json() (*mtpy.core.mt\_location.MTLocation* method), 140  
from\_json() (*mtpy.core.MTLocation* method), 147  
from\_modem() (*mtpy.core.mt\_data.MTData* method), 127  
from\_modem() (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D* method), 311  
from\_modem() (*mtpy.modeling.StructuredGrid3D* method), 331  
from\_modem() (*mtpy.MTData* method), 410  
from\_modem\_data() (*mtpy.core.mt\_data.MTData* method), 127

```

from_modem_data() (mtpy.MTData method), 410
from_mt_data() (mtpy.core.mt_collection.MTCollection
 method), 119
from_mt_data() (mtpy.MTCollection method), 401
from_mt_dataframe() (mtpy.core.mt_data.MTData
 method), 128
from_mt_dataframe() (mtpy.MTData method), 410
from_mth5s() (mtpy.processing.run_summary.RunSummary
 method), 357
from_mth5s() (mtpy.processing.RunSummary method),
 376
from_occam2d() (mtpy.core.mt_data.MTData method),
 128
from_occam2d() (mtpy.MTData method), 410
from_occam2d_data() (mtpy.core.mt_data.MTData
 method), 128
from_occam2d_data() (mtpy.MTData method), 411
from_run_summary() (mtpy.processing.kernel_dataset.KernelDataset
 method), 352
from_run_summary() (mtpy.processing.KernelDataset
 method), 373
from_t_object() (mtpy.core.mt_dataframe.MTDataFrame
 method), 138
from_t_object() (mtpy.core.MTDataFrame method),
 145
from_wl_write_station_file()
 (mtpy.modeling.ws3dinv.stations.WSStation
 method), 296
from_wl_write_station_file()
 (mtpy.modeling.ws3dinv.WSStation method),
 302
from_ws3dinv() (mtpy.modeling.structured_mesh_3d.StructuredGrid3D
 method), 312
from_ws3dinv() (mtpy.modeling.StructuredGrid3D
 method), 332
from_ws3dinv_initial()
 (mtpy.modeling.structured_mesh_3d.StructuredGrid3D
 method), 312
from_ws3dinv_initial()
 (mtpy.modeling.StructuredGrid3D method), 332
from_xarray() (mtpy.core.transfer_function.base.TFBase
 method), 91
from_z_object() (mtpy.core.mt_dataframe.MTDataFrame
 method), 138
from_z_object() (mtpy.core.MTDataFrame method),
 145
generate_profile() (mtpy.core.mt_stations.MTStations
 method), 142
generate_profile() (mtpy.core.MTStations method),
 149
generate_profile_from_strike()
 (mtpy.core.mt_stations.MTStations method),
 142
generate_profile_from_strike()
 (mtpy.core.MTStations method), 149
get_all_edi_files()
 (mtpy.utils.edi_folders.EdiFolders method),
 383
get_birrp_config_fn() (mtpy.processing.birrp.J2Edi
 method), 342
get_color() (in module mtpy.imaging.mtcOLORS), 181
get_color_map() (mtpy.imaging.mplplot_tools.ellipses.MTEllipse
 method), 166
get_color_map() (mtpy.imaging.mplplot_tools.MTEllipse
 method), 174
get_elevation_from_national_map()
 (mtpy.core.mt_location.MTLlocation method),
 140
get_elevation_from_national_map()
 (mtpy.core.MTLlocation method), 147
get_estimate() (mtpy.imaging.plot_strike.PlotStrike
 method), 195
get_estimate() (mtpy.imaging.PlotStrike method), 205
get_filename() (in module mtpy.utils.filehandling),
 386
get_interp1d_functions_t()
 (mtpy.imaging.mplplot_tools.base.PlotBaseMaps
 static method), 164
get_interp1d_functions_t()
 (mtpy.imaging.mplplot_tools.PlotBaseMaps
 static method), 176
get_interp1d_functions_z()
 (mtpy.imaging.mplplot_tools.base.PlotBaseMaps
 static method), 164
get_interp1d_functions_z()
 (mtpy.imaging.mplplot_tools.PlotBaseMaps
 static method), 176
get_j_file() (mtpy.processing.birrp.J2Edi method),
 342
get_latlon_extents_from_modem_data() (in mod-
 ule mtpy.utils.basemap_tools), 377
get_log_tick_labels()
 (in module mtpy.imaging.mplplot_tools), 178
get_log_tick_labels()
 (in module mtpy.imaging.mplplot_tools.utils), 172
get_lower_left_corner()
 (mtpy.modeling.structured_mesh_3d.StructuredGrid3D
 method), 312
get_lower_left_corner()
 (mtpy.modeling.StructuredGrid3D method),
 332
get_mean() (mtpy.imaging.plot_strike.PlotStrike
 method), 332

```

## G

gausswin() (in module mtpy.processing.tf), 359  
gen\_hist\_bins() (in module mtpy.utils.matplotlib\_utils), 391  
generate\_profile() (mtpy.core.mt\_stations.MTStations
 method), 142

get\_latlon\_extents\_from\_modem\_data() (in mod-
 ule mtpy.utils.basemap\_tools), 377  
get\_log\_tick\_labels()
 (in module mtpy.imaging.mplplot\_tools), 178  
get\_log\_tick\_labels()
 (in module mtpy.imaging.mplplot\_tools.utils), 172  
get\_lower\_left\_corner()
 (mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D
 method), 312  
get\_lower\_left\_corner()
 (mtpy.modeling.StructuredGrid3D method),
 332  
get\_mean() (mtpy.imaging.plot\_strike.PlotStrike
 method), 332

method), 195  
get\_mean() (mtpy.imaging.PlotStrike method), 205  
get\_median() (mtpy.imaging.plot\_strike.PlotStrike method), 195  
get\_median() (mtpy.imaging.PlotStrike method), 205  
get\_misfit() (mtpy.modeling.winglink.PlotMisfitPseudoSection.get\_topo\_array() (mtpy.imaging.PlotStrike method), 205  
get\_mode() (mtpy.imaging.plot\_strike.PlotStrike method), 195  
get\_mode() (mtpy.imaging.PlotStrike method), 205  
get\_n\_stations() (mtpy.modeling.modem.Data method), 226  
get\_n\_stations() (mtpy.modeling.modem.data.Data method), 210  
get\_n\_stations() (mtpy.modeling.ws3dinv.data.WSData method), 294  
get\_n\_stations() (mtpy.modeling.ws3dinv.WSData method), 300  
get\_nearby\_stations() (mtpy.core.mt\_data.MTData method), 128  
get\_nearby\_stations() (mtpy.MTData method), 411  
get\_nearest\_index() (in module mtpy.modeling.mesh\_tools), 305  
get\_next\_fig\_num() (in module mtpy.utils.matplotlib\_utils), 391  
get\_num\_free\_params() (mtpy.modeling.occam2d.Regularization method), 281  
get\_num\_free\_params() (mtpy.modeling.occam2d.regularization.Regularizer method), 269  
get\_padding\_cells() (in module mtpy.modeling.mesh\_tools), 305  
get\_padding\_cells2() (in module mtpy.modeling.mesh\_tools), 305  
get\_padding\_from\_stretch() (in module mtpy.modeling.mesh\_tools), 306  
get\_parameters() (mtpy.modeling.modem.convariance.Covariance method), 208  
get\_parameters() (mtpy.modeling.modem.Covariance method), 225  
get\_pathlist() (in module mtpy.utils.filehandling), 386  
get\_period() (mtpy.core.mt\_dataframe.MTDataFrame method), 138  
get\_period() (mtpy.core.MTDataFrame method), 145  
get\_period\_df() (mtpy.modeling.ws3dinv.data.WSData method), 294  
get\_period\_df() (mtpy.modeling.ws3dinv.WSData method), 300  
get\_period\_limits() (in module mtpy.imaging.mtplot\_tools.utils), 172  
get\_period\_list() (in module mtpy.utils.calculator), 378  
get\_periods() (mtpy.core.mt\_data.MTData method), 128  
get\_periods() (mtpy.MTData method), 411  
get\_plot\_array() (mtpy.imaging.plot\_strike.PlotStrike method), 195  
get\_topo\_array() (mtpy.imaging.PlotStrike method), 205  
get\_plot\_color() (in module mtpy.imaging.mtcOLORS), 181  
get\_plot\_xy() (in module mtpy.imaging.mtplot\_tools.map\_interpolation\_tools), 166  
get\_profile() (mtpy.core.mt\_data.MTData method), 129  
get\_profile() (mtpy.MTData method), 411  
get\_pt\_color\_array() (mtpy.imaging.mtplot\_tools.ellipses.MTEllipse method), 166  
get\_pt\_color\_array() (mtpy.imaging.mtplot\_tools.MTEllipse method), 175  
get\_pt\_color\_array() (mtpy.imaging.plot\_residual\_pt\_ps.PlotResidualPTPseudoSection.get\_range() (mtpy.imaging.mtplot\_tools.ellipses.MTEllipse method), 166  
get\_pt\_color\_array() (mtpy.imaging.PlotResidualPTPseudoSection method), 203  
get\_range() (mtpy.imaging.mtplot\_tools.ellipses.MTEllipse method), 166  
get\_range() (mtpy.imaging.mtplot\_tools.MTEllipse method), 175  
get\_rotation\_matrix() (in module mtpy.utils.calculator), 378  
get\_rounding() (in module mtpy.modeling.mesh\_tools), 306  
get\_run\_object() (mtpy.processing.kernel\_dataset.KernelDataset method), 352  
get\_run\_object() (mtpy.processing.KernelDataset method), 373  
get\_run\_summary() (mtpy.processing.base.BaseProcessing method), 341  
get\_sampling\_interval\_fromdatafile() (in module mtpy.utils.filehandling), 387  
get\_station() (mtpy.core.mt\_data.MTData method), 129  
get\_station() (mtpy.MTData method), 411  
get\_station\_buffer() (in module mtpy.modeling.mesh\_tools), 306  
get\_station\_df() (mtpy.core.mt\_dataframe.MTDataFrame method), 138  
get\_station\_df() (mtpy.core.MTDataFrame method), 145  
get\_station\_distances() (mtpy.core.mt\_dataframe.MTDataFrame

*method), 138*  
**get\_station\_distances()** (*mtpy.core.MTDataFrame method*), 145  
**get\_station\_locations()**  
*(mtpy.modeling.modem.station.Stations method), 221*  
**get\_station\_metadata()**  
*(mtpy.processing.kernel\_dataset.KernelDataset method), 352*  
**get\_station\_metadata()**  
*(mtpy.processing.KernelDataset method), 373*  
**get\_stats()** (*mtpy.imaging.plot\_strike.PlotStrike method*), 195  
**get\_stats()** (*mtpy.imaging.PlotStrike method*), 205  
**get\_subset()** (*mtpy.core.mt\_data.MTData method*), 129  
**get\_subset()** (*mtpy.MTData method*), 412  
**get\_survey()** (*mtpy.core.mt\_data.MTData method*), 130  
**get\_survey()** (*mtpy.MTData method*), 412  
**get\_tf()** (*mtpy.core.mt\_collection.MTCollection method*), 119  
**get\_tf()** (*mtpy.MTCollection method*), 402  
**get\_ts\_header\_string()** (*in module mtpy.utils.filehandling*), 387  
**GISError**, 389  
**grid\_centre()** (*in module mtpy.modeling.mesh\_tools*), 306  
**griddata\_interpolate()** (*in module mtpy.imaging.mtplot\_tools*), 178  
**griddata\_interpolate()** (*in module mtpy.imaging.mtplot\_tools.map\_interpolation\_tools*), 166

**H**

**has\_data()** (*mtpy.core.mt\_collection.MTCollection method*), 119  
**has\_data()** (*mtpy.MTCollection method*), 402  
**has\_impedance()** (*mtpy.core.transfer\_function.z\_analysis.ZInvariant method*), 89  
**has\_impendance()** (*mtpy.core.transfer\_function.z\_analysis.zinvariants.ZInvariants method*), 88  
**has\_kernel\_dataset()**  
*(mtpy.processing.base.BaseProcessing method), 341*  
**has\_local\_mth5()** (*mtpy.processing.kernel\_dataset.KernelDataset method*), 352  
**has\_local\_mth5()** (*mtpy.processing.KernelDataset method*), 373  
**has\_remote\_mth5()** (*mtpy.processing.kernel\_dataset.KernelDataset method*), 353  
**has\_remote\_mth5()** (*mtpy.processing.KernelDataset method*), 373

**has\_run\_summary()** (*mtpy.processing.base.BaseProcessing method*), 341  
**hx\_metadata** (*mtpy.core.mt.MT property*), 114  
**hx\_metadata** (*mtpy.MT property*), 396  
**hy\_metadata** (*mtpy.core.mt.MT property*), 114  
**hy\_metadata** (*mtpy.MT property*), 396  
**hz\_metadata** (*mtpy.core.mt.MT property*), 114  
**hz\_metadata** (*mtpy.MT property*), 396

|

**impedance** (*mtpy.core.mt\_dataframe.MTDataFrame property*), 138  
**impedance** (*mtpy.core.MTDataFrame property*), 145  
**impedance\_units** (*mtpy.core.mt.MT property*), 114  
**impedance\_units** (*mtpy.core.mt\_data.MTData property*), 130  
**impedance\_units** (*mtpy.MT property*), 396  
**impedance\_units** (*mtpy.MTData property*), 412  
**in\_hull()** (*in module mtpy.imaging.mtplot\_tools.map\_interpolation\_tools*), 166  
**initialise\_basemap()** (*in module mtpy.utils.basemap\_tools*), 377  
**initialize\_dataframe\_for\_processing()**  
*(mtpy.processing.kernel\_dataset.KernelDataset method), 353*  
**initialize\_dataframe\_for\_processing()**  
*(mtpy.processing.KernelDataset method), 373*  
**initialize\_mth5s()** (*mtpy.processing.kernel\_dataset.KernelDataset method*), 353  
**initialize\_mth5s()** (*mtpy.processing.KernelDataset method*), 374  
**input\_channels** (*mtpy.processing.kernel\_dataset.KernelDataset property*), 353  
**input\_channels** (*mtpy.processing.KernelDataset property*), 374  
**interpolate()** (*mtpy.core.mt.MT method*), 114  
**interpolate()** (*mtpy.core.mt\_data.MTData method*), 412

*invariant\_zinvariants.ZInvariants*  
**interpolate()** (*mtpy.core.transfer\_function.base.TFBase method*), 214  
**interpolate()** (*mtpy.MT method*), 396  
**interpolate()** (*mtpy.MTData method*), 412  
**interpolate\_elevation()**  
*(mtpy.modeling.modem.Model method), 230*  
**interpolate\_elevation()**  
*(mtpy.modeling.modem.model.Model method), 214*  
**interpolate\_elevation()**  
*(mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D method), 313*

interpolate\_elevation()  
    (*mtpy.modeling.StructuredGrid3D* method), 333

interpolate\_elevation\_to\_grid() (in module *mtpy.modeling.mesh\_tools*), 306

interpolate\_to\_even\_grid()  
    (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D* method), 313

interpolate\_to\_even\_grid()  
    (*mtpy.modeling.StructuredGrid3D* method), 333

interpolate\_to\_map() (in module *mtpy.imaging.mtplot\_tools*), 178

interpolate\_to\_map() (in module *mtpy.imaging.mtplot\_tools.map\_interpolation\_tools*), 166

interpolate\_to\_map()  
    (*mtpy.imaging.mtplot\_tools.base.PlotBaseMaps* method), 164

interpolate\_to\_map()  
    (*mtpy.imaging.mtplot\_tools.PlotBaseMaps* method), 176

interpolate\_to\_map\_griddata() (in module *mtpy.imaging.mtplot\_tools.map\_interpolation\_tools*), 167

interpolate\_to\_map\_triangulate() (in module *mtpy.imaging.mtplot\_tools.map\_interpolation\_tools*), 167

intervals\_overlap() (in module *mtpy.processing.kernel\_dataset*), 355

invariants (*mtpy.core.transfer\_function.Z* property), 107

invariants (*mtpy.core.transfer\_function.zZ* property), 98

invariants (*mtpy.core.Z* property), 157

inverse (*mtpy.core.transfer\_function.base.TFBase* property), 92

inverse\_problem (*mtpy.modeling.simpeg.recipes.Simpeg2D* property), 288

invertmatrix\_incl\_errors() (in module *mtpy.utils.calculator*), 378

is\_single\_station (*mtpy.processing.kernel\_dataset*.KernelDataset property), 353

is\_single\_station (*mtpy.processing*.KernelDataset property), 374

iterations (*mtpy.modeling.simpeg.recipes.Simpeg2D* property), 288

**J**

J2Edi (class in *mtpy.processing.birrp*), 342

**K**

KernelDataset (class in *mtpy.processing*), 371

KernelDataset (class in *mtpy.processing.kernel\_dataset*), 350

**L**

lat (*mtpy.modeling.modem.station.Stations* property), 221

latitude (*mtpy.core.mt\_dataframe.MTDataFrame* property), 138

latitude (*mtpy.core.mt\_location.MTLocation* property), 140

latitude (*mtpy.core.MTDataFrame* property), 145

latitude (*mtpy.core.MTLocation* property), 147

local\_df (*mtpy.processing.kernel\_dataset*.KernelDataset property), 353

local\_df (*mtpy.processing*.KernelDataset property), 374

local\_mth5\_path (*mtpy.processing.kernel\_dataset*.KernelDataset property), 353

local\_mth5\_path (*mtpy.processing*.KernelDataset property), 374

local\_station\_id (*mtpy.processing.kernel\_dataset*.KernelDataset property), 353

local\_station\_id (*mtpy.processing*.KernelDataset property), 374

local\_survey\_id (*mtpy.processing.kernel\_dataset*.KernelDataset property), 353

local\_survey\_id (*mtpy.processing*.KernelDataset property), 374

local\_survey\_metadata  
    (*mtpy.processing.kernel\_dataset*.KernelDataset property), 353

local\_survey\_metadata  
    (*mtpy.processing*.KernelDataset property), 374

locate\_bad\_phase\_points()  
    (*mtpy.utils.estimate\_tf\_quality\_factor.EMTFStats* method), 384

locate\_bad\_res\_points()  
    (*mtpy.utils.estimate\_tf\_quality\_factor.EMTFStats* method), 384

locate\_bad\_tipper\_points()  
    (*mtpy.utils.estimate\_tf\_quality\_factor.EMTFStats* method), 384

lon (*mtpy.modeling.modem.station.Stations* property), 221

longitude (*mtpy.core.mt\_dataframe.MTDataFrame* property), 138

longitude (*mtpy.core.mt\_location.MTLocation* property), 140

longitude (*mtpy.core.MTDataFrame* property), 146

longitude (*mtpy.core.MTLocation* property), 147

low\_pass() (in module *mtpy.processing.filter*), 348

# M

`mag_error (mtpy.core.Tipper property)`, 155  
`mag_error (mtpy.core.transfer_function.Tipper property)`, 105  
`mag_error (mtpy.core.transfer_function.tipper.Tipper property)`, 96  
`mag_imag (mtpy.core.Tipper property)`, 155  
`mag_imag (mtpy.core.transfer_function.Tipper property)`, 105  
`mag_imag (mtpy.core.transfer_function.tipper.Tipper property)`, 96  
`mag_model_error (mtpy.core.Tipper property)`, 155  
`mag_model_error (mtpy.core.transfer_function.Tipper property)`, 105  
`mag_model_error (mtpy.core.transfer_function.tipper.Tipper property)`, 96  
`mag_real (mtpy.core.Tipper property)`, 155  
`mag_real (mtpy.core.transfer_function.Tipper property)`, 105  
`mag_real (mtpy.core.transfer_function.tipper.Tipper property)`, 96  
`make_color_list() (in module mtpy.imaging.mtplot_tools.utils)`, 172  
`make_file_list() (mtpy.core.mt_collection.MTCollection static method)`, 119  
`make_file_list() (mtpy.MTCollection static method)`, 402  
`make_fn_lines_block_00() (mtpy.processing.birrp.ScriptFile method)`, 346  
`make_fn_lines_block_n() (mtpy.processing.birrp.ScriptFile method)`, 346  
`make_log_increasing_array() (in module mtpy.modeling.mesh_tools)`, 307  
`make_log_increasing_array() (in module mtpy.utils.calculator)`, 378  
`make_mesh() (mtpy.modeling.modem.Model method)`, 231  
`make_mesh() (mtpy.modeling.modem.model.Model method)`, 215  
`make_mesh() (mtpy.modeling.simpeg.make_2d_mesh.QuadTreeMesh method)`, 290  
`make_mesh() (mtpy.modeling.simpeg.make_2d_mesh.StructuredMesh method)`, 291  
`make_mesh() (mtpy.modeling.simpeg.recipes.Simpeg2D method)`, 288  
`make_mesh() (mtpy.modeling.structured_mesh_3d.StructuredGrid3D method)`, 313  
`make_mesh() (mtpy.modeling.StructuredGrid3D method)`, 333  
`make_pt_cb() (mtpy.imaging.mtplot_tools.plot_settings.PlotSettings method)`, 169  
`make_pt_cb() (mtpy.imaging.mtplot_tools.PlotSettings method)`, 370  
`method), 177`  
`make_shp_files() (mtpy.gis.shapefile_creator.ShapefileCreator method)`, 161  
`make_strike_df() (mtpy.imaging.plot_strike.PlotStrike method)`, 195  
`make_strike_df() (mtpy.imaging.PlotStrike method)`, 205  
`make_unique_filename() (in module mtpy.utils.filehandling)`, 387  
`make_unique_folder() (in module mtpy.utils.filehandling)`, 387  
`make_value_str() (in module mtpy.imaging.mtplot_tools.utils)`, 172  
`make_z_mesh() (mtpy.modeling.modem.Model method)`, 231  
`make_z_mesh() (mtpy.modeling.modem.model.Model method)`, 215  
`make_z_mesh() (mtpy.modeling.structured_mesh_3d.StructuredGrid3D method)`, 314  
`make_z_mesh() (mtpy.modeling.StructuredGrid3D method)`, 334  
`map_scale (mtpy.imaging.plot_phase_tensor_maps.PlotPhaseTensorMaps property)`, 184  
`map_scale (mtpy.imaging.plot_residual_pt_maps.PlotResidualPTMaps property)`, 188  
`map_scale (mtpy.imaging.PlotPhaseTensorMaps property)`, 198  
`map_scale (mtpy.imaging.PlotResidualPTMaps property)`, 201  
`map_units (mtpy.imaging.plot_resphase_maps.PlotResPhaseMaps property)`, 191  
`map_units (mtpy.imaging.PlotResPhaseMaps property)`, 200  
`mask_from_datafile() (mtpy.modeling.occam2d.data.Occam2DDData method)`, 261  
`mask_from_datafile() (mtpy.modeling.occam2d.Occam2DDData method)`, 277  
`mask_zeros() (mtpy.modeling.errors.ModelErrors method)`, 304  
`master_dataframe (mtpy.core.mt_collection.MTCollection property)`, 120  
`master_dataframe (mtpy.MTCollection property)`, 402  
`measurement_error (mtpy.modeling.errors.ModelErrors property)`, 304  
`merge_transfer_functions()`  
`(mtpy.processing.aurora.AuroraProcessing method)`, 339  
`merge_transfer_functions()`  
`(mtpy.processing.AuroraProcessing method)`, 370  
`Mesh (class in mtpy.modeling.occam2d)`, 272  
`Mesh (class in mtpy.modeling.occam2d.mesh)`, 262

mesh (*mtpy.modeling.simpeg.recipes.inversion\_1d.Simpeg1D*.**model\_epsg** (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D* property), 285  
mesh (*mtpy.modeling.simpeg.recipes.Simpeg1D* property), 286  
**mini\_summary** (*mtpy.processing.kernel\_dataset.KernelDataset* property), 353  
**mini\_summary** (*mtpy.processing.KernelDataset* property), 374  
**mini\_summary** (*mtpy.processing.run\_summary.RunSummary* property), 357  
**mini\_summary** (*mtpy.processing.RunSummary* property), 376  
**mode** (*mtpy.modeling.errors.ModelErrors* property), 304  
**mode** (*mtpy.modeling.occam1d.data.Occam1DData* property), 238  
**mode** (*mtpy.modeling.occam1d.Occam1DData* property), 250  
**mode** (*mtpy.modeling.simpeg.recipes.inversion\_1d.Simpeg1D*.**Model** property), 285  
**mode** (*mtpy.modeling.simpeg.recipes.Simpeg1D* property), 287  
**mode\_01** (*mtpy.modeling.occam1d.data.Occam1DData* property), 238  
**mode\_01** (*mtpy.modeling.occam1d.Occam1DData* property), 250  
**mode\_02** (*mtpy.modeling.occam1d.data.Occam1DData* property), 238  
**mode\_02** (*mtpy.modeling.occam1d.Occam1DData* property), 250  
**Model** (class in *mtpy.modeling.modem*), 227  
**Model** (class in *mtpy.modeling.modem.model*), 211  
**model\_east** (*mtpy.core.mt\_dataframe.MTDataFrame* property), 138  
**model\_east** (*mtpy.core.mt\_location.MTLocation* property), 140  
**model\_east** (*mtpy.core.MTDataFrame* property), 146  
**model\_east** (*mtpy.core.MTLocation* property), 147  
**model\_elevation** (*mtpy.core.mt\_dataframe.MTDataFrame* property), 138  
**model\_elevation** (*mtpy.core.mt\_location.MTLocation* property), 140  
**model\_elevation** (*mtpy.core.MTDataFrame* property), 146  
**model\_elevation** (*mtpy.core.MTLocation* property), 147  
**model\_epsg** (*mtpy.core.mt\_stations.MTStations* property), 142  
**model\_epsg** (*mtpy.core.MTStations* property), 149  
**model\_epsg** (*mtpy.modeling.modem.Model* property), 231  
**model\_epsg** (*mtpy.modeling.modem.model.Model* property), 215  
**model\_epsg** (*mtpy.modeling.modem.station.Stations* property), 222  
**model\_fn** (*mtpy.modeling.modem.Model* property), 231  
**model\_fn** (*mtpy.modeling.modem.model.Model* property), 215  
**model\_fn** (*mtpy.modeling.occam1d.Occam1DStartup* property), 255  
**model\_fn** (*mtpy.modeling.occam1d.startup.Occam1DStartup* property), 247  
**model\_fn** (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D* property), 314  
**model\_fn** (*mtpy.modeling.StructuredGrid3D* property), 334  
**model\_north** (*mtpy.core.mt\_dataframe.MTDataFrame* property), 138  
**model\_north** (*mtpy.core.mt\_location.MTLocation* property), 140  
**model\_north** (*mtpy.core.MTDataFrame* property), 146  
**model\_north** (*mtpy.core.MTLocation* property), 147  
**model\_parameters** (*mtpy.modeling.modem.Data* property), 226  
**model\_parameters** (*mtpy.modeling.modem.data.Data* property), 210  
**model\_parameters** (*mtpy.modeling.modem.Model* property), 231  
**model\_parameters** (*mtpy.modeling.modem.model.Model* property), 215  
**model\_parameters** (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D* property), 314  
**model\_parameters** (*mtpy.modeling.StructuredGrid3D* property), 334  
**model\_utm\_zone** (*mtpy.modeling.modem.station.Stations* property), 222  
**ModelError** (class in *mtpy.modeling.errors*), 303  
**ModEMConfig** (class in *mtpy.modeling.modem*), 226  
**ModEMConfig** (class in *mtpy.modeling.modem.config*), 206  
**ModEMError**, 210, 227  
**modifiedb()** (in module *mtpy.processing.tf*), 359  
**module**  
    MT, 110  
    mtpy, 393  
    mtpy.analysis, 84  
    mtpy.analysis.residual\_phase\_tensor, 83  
    mtpy.core, 145  
    mtpy.core.mt, 110  
    mtpy.core.mt\_collection, 117  
    mtpy.core.mt\_data, 124  
    mtpy.core.mt\_dataframe, 137  
    mtpy.core.mt\_location, 139  
    mtpy.core.mt\_stations, 141  
    mtpy.core.transfer\_function, 102

mtpy.core.transfer\_function.base, 91  
 mtpy.core.transfer\_function.pt, 93  
 mtpy.core.transfer\_function.tipper, 96  
 mtpy.core.transfer\_function.z, 97  
 mtpy.core.transfer\_function.z\_analysis,  
     88  
 mtpy.core.transfer\_function.z\_analysis.distortion, 84  
     85  
 mtpy.core.transfer\_function.z\_analysis.niblett, 85  
     86  
 mtpy.core.transfer\_function.z\_analysis.zinvariance, 87  
     88  
 mtpy.gis, 162  
 mtpy.gis.raster\_tools, 160  
 mtpy.gis.shapefile\_creator, 161  
 mtpy.imaging, 196  
 mtpy.imaging.mtcolors, 181  
 mtpy.imaging.mtplot\_tools, 173  
 mtpy.imaging.mtplot\_tools.arrows, 162  
 mtpy.imaging.mtplot\_tools.base, 163  
 mtpy.imaging.mtplot\_tools.ellipses, 165  
 mtpy.imaging.mtplot\_tools.map\_interpolation, 166  
     166  
 mtpy.imaging.mtplot\_tools.plot\_settings,  
     169  
 mtpy.imaging.mtplot\_tools.plotters, 170  
 mtpy.imaging.mtplot\_tools.utils, 172  
 mtpy.imaging.plot\_mt\_response, 181  
 mtpy.imaging.plot\_mt\_responses, 182  
 mtpy.imaging.plot\_penetration\_depth\_1d,  
     183  
 mtpy.imaging.plot\_penetration\_depth\_map,  
     183  
 mtpy.imaging.plot\_phase\_tensor\_maps, 184  
 mtpy.imaging.plot\_phase\_tensor\_pseudosection,  
     185  
 mtpy.imaging.plot\_pseudosection, 186  
 mtpy.imaging.plot\_pt, 186  
 mtpy.imaging.plot\_residual\_pt\_maps, 186  
 mtpy.imaging.plot\_residual\_pt\_ps, 188  
 mtpy.imaging.plot\_resphase\_maps, 191  
 mtpy.imaging.plot\_spectrogram, 192  
 mtpy.imaging.plot\_stations, 193  
 mtpy.imaging.plot\_strike, 193  
 mtpy.modeling, 327  
 mtpy.modeling.errors, 303  
 mtpy.modeling.gocad, 305  
 mtpy.modeling.mesh\_tools, 305  
 mtpy.modeling.modem, 223  
 mtpy.modeling.modem.config, 206  
 mtpy.modeling.modem.control\_fwd, 206  
 mtpy.modeling.modem.control\_inv, 207  
 mtpy.modeling.modem.convariance, 208  
 mtpy.modeling.modem.data, 209  
     210  
     211  
     219  
     221  
     249  
     237  
     240  
     243  
     244  
     247  
     272  
     260  
     262  
     266  
     268  
     270  
     284  
     283  
     283  
     292  
     290  
     286  
     285  
     307  
     319  
     326  
     297  
     292  
     295  
     296  
     393  
     370  
     341  
     339  
     341  
     341  
     347  
     350  
     356  
     358  
     393  
     377  
     378  
     380  
     381  
     383  
     383  
     385  
     386  
     389

`mtpy.utils.matplotlib_utils`, 391  
`mtpy.utils.mtpy_decorator`, 391  
`mtpy.utils.plot_rms_iterations`, 391  
`mtpy.utils.sensor_orientation_correction`, 392  
    392  
`TFBase`, 91  
`MT`  
    module, 110  
`MT (class in mtpy)`, 393  
`MT (class in mtpy.core.mt)`, 110  
`mt_dataframe (mtpy.gis.shapefile_creator.ShapefileCreator property)`, 162  
`mt_list (mtpy.core.mt_data.MTData property)`, 130  
`mt_list (mtpy.MTData property)`, 413  
`MTArrows (class in mtpy.imaging.mtplot_tools)`, 173  
`MTArrows (class in mtpy.imaging.mtplot_tools.arrows)`, 162  
`MTCollection (class in mtpy)`, 400  
`MTCollection (class in mtpy.core.mt_collection)`, 117  
`MTData (class in mtpy)`, 406  
`MTData (class in mtpy.core.mt_data)`, 124  
`MTDataFrame (class in mtpy.core)`, 145  
`MTDataFrame (class in mtpy.core.mt_dataframe)`, 137  
`MTEllipse (class in mtpy.imaging.mtplot_tools)`, 173  
`MTEllipse (class in mtpy.imaging.mtplot_tools.ellipses)`, 165  
`mth5_filename (mtpy.core.mt_collection.MTCollection property)`, 120  
`mth5_filename (mtpy.MTCollection property)`, 402  
`mth5_list (mtpy.processing.base.BaseProcessing property)`, 341  
`mth5_objs (mtpy.processing.kernel_datasetKernelDataset property)`, 354  
`mth5_objs (mtpy.processing.KernelDataset property)`, 374  
`MTLocation (class in mtpy.core)`, 146  
`MTLocation (class in mtpy.core.mt_location)`, 139  
`mtpy`  
    module, 393  
`mtpy.analysis`  
    module, 84  
`mtpy.analysis.residual_phase_tensor`  
    module, 83  
`mtpy.core`  
    module, 145  
`mtpy.core.mt`  
    module, 110  
`mtpy.core.mt_collection`  
    module, 117  
`mtpy.core.mt_data`  
    module, 124  
`mtpy.core.mt_dataframe`  
    module, 137  
`mtpy.core.mt_location`

`module`, 139  
`mtpy.core.mt_stations`  
    module, 141  
`mtpy.core.transfer_function`  
    module, 102  
`mtpy.core.transfer_function.base`  
    module, 91  
`mtpy.core.transfer_function.pt`  
    module, 93  
`mtpy.core.transfer_function.tipper`  
    module, 96  
`mtpy.core.transfer_function.z`  
    module, 97  
`mtpy.core.transfer_function.z_analysis`  
    module, 88  
`mtpy.core.transfer_function.z_analysis.distortion`  
    module, 84  
`mtpy.core.transfer_function.z_analysis.niblettbostick`  
    module, 85  
`mtpy.core.transfer_function.z_analysis.zinvariants`  
    module, 87  
`mtpy.gis`  
    module, 162  
`mtpy.gis.raster_tools`  
    module, 160  
`mtpy.gis.shapefile_creator`  
    module, 161  
`mtpy.imaging`  
    module, 196  
`mtpy.imaging.mtcolors`  
    module, 181  
`mtpy.imaging.mtplot_tools`  
    module, 173  
`mtpy.imaging.mtplot_tools.arrows`  
    module, 162  
`mtpy.imaging.mtplot_tools.base`  
    module, 163  
`mtpy.imaging.mtplot_tools.ellipses`  
    module, 165  
`mtpy.imaging.mtplot_tools.map_interpolation_tools`  
    module, 166  
`mtpy.imaging.mtplot_tools.plot_settings`  
    module, 169  
`mtpy.imaging.mtplot_tools.plotters`  
    module, 170  
`mtpy.imaging.mtplot_tools.utils`  
    module, 172  
`mtpy.imaging.plot_mt_response`  
    module, 181  
`mtpy.imaging.plot_mt_responses`  
    module, 182  
`mtpy.imaging.plot_penetration_depth_1d`  
    module, 183  
`mtpy.imaging.plot_penetration_depth_map`

```

 module, 183
mtpy.imaging.plot_phase_tensor_maps
 module, 184
mtpy.imaging.plot_phase_tensor_pseudosection
 module, 185
mtpy.imaging.plot_pseudosection
 module, 186
mtpy.imaging.plot_pt
 module, 186
mtpy.imaging.plot_residual_pt_maps
 module, 186
mtpy.imaging.plot_residual_pt_ps
 module, 188
mtpy.imaging.plot_resphase_maps
 module, 191
mtpy.imaging.plot_spectrogram
 module, 192
mtpy.imaging.plot_stations
 module, 193
mtpy.imaging.plot_strike
 module, 193
mtpy.modeling
 module, 327
mtpy.modeling.errors
 module, 303
mtpy.modeling.gocad
 module, 305
mtpy.modeling.mesh_tools
 module, 305
mtpy.modeling.modem
 module, 223
mtpy.modeling.modem.config
 module, 206
mtpy.modeling.modem.control_fwd
 module, 206
mtpy.modeling.modem.control_inv
 module, 207
mtpy.modeling.modem.convariance
 module, 208
mtpy.modeling.modem.data
 module, 209
mtpy.modeling.modem.exception
 module, 210
mtpy.modeling.modem.model
 module, 211
mtpy.modeling.modem.residual
 module, 219
mtpy.modeling.modem.station
 module, 221
mtpy.modeling.occam1d
 module, 249
mtpy.modeling.occam1d.data
 module, 237
mtpy.modeling.occam1d.model
 module, 240
mtpy.modeling.occam1d.plot_12
 module, 243
mtpy.modeling.occam1d.plot_response
 module, 244
mtpy.modeling.occam1d.run
 module, 247
mtpy.modeling.occam1d.startup
 module, 247
mtpy.modeling.occam2d
 module, 272
mtpy.modeling.occam2d.data
 module, 260
mtpy.modeling.occam2d.mesh
 module, 262
mtpy.modeling.occam2d.model
 module, 266
mtpy.modeling.occam2d.regularization
 module, 268
mtpy.modeling.occam2d.startup
 module, 270
mtpy.modeling.plots
 module, 284
mtpy.modeling.plots.plot_mesh
 module, 283
mtpy.modeling.plots.plot_modem_rms
 module, 283
mtpy.modeling.simpeg
 module, 292
mtpy.modeling.simpeg.make_2d_mesh
 module, 290
mtpy.modeling.simpeg.recipes
 module, 286
mtpy.modeling.simpeg.recipes.inversion_1d
 module, 285
mtpy.modeling.structured_mesh_3d
 module, 307
mtpy.modeling.winglink
 module, 319
mtpy.modeling.winglinktools
 module, 326
mtpy.modeling.ws3dinv
 module, 297
mtpy.modeling.ws3dinv.data
 module, 292
mtpy.modeling.ws3dinv.startup
 module, 295
mtpy.modeling.ws3dinv.stations
 module, 296
mtpy.mtpy_globals
 module, 393
mtpy.processing
 module, 370
mtpy.processing.aurora

```

module, 341  
mtpy.processing.aurora.process\_aurora  
    module, 339  
mtpy.processing.base  
    module, 341  
mtpy.processing.birrp  
    module, 341  
mtpy.processing.filter  
    module, 347  
mtpy.processing.kernel\_dataset  
    module, 350  
mtpy.processing.run\_summary  
    module, 356  
mtpy.processing.tf  
    module, 358  
mtpy.utils  
    module, 393  
mtpy.utils.basemap\_tools  
    module, 377  
mtpy.utils.calculator  
    module, 378  
mtpy.utils.concatenate\_input  
    module, 380  
mtpy.utils.configfile  
    module, 381  
mtpy.utils.edi\_folders  
    module, 383  
mtpy.utils.estimate\_tf\_quality\_factor  
    module, 383  
mtpy.utils.exceptions  
    module, 385  
mtpy.utils.filehandling  
    module, 386  
mtpy.utils.gis\_tools  
    module, 389  
mtpy.utils.matplotlib\_utils  
    module, 391  
mtpy.utils.mtpy\_decorator  
    module, 391  
mtpy.utils.plot\_rms\_iterations  
    module, 391  
mtpy.utils.sensor\_orientation\_correction  
    module, 392  
MTpyError\_config\_file, 385  
MTpyError\_EDI, 385  
MTpyError\_edi\_file, 385  
MTpyError\_file\_handling, 385  
MTpyError\_float, 385  
MTpyError\_input\_arguments, 385  
MTpyError\_module\_import, 385  
MTpyError\_occam, 385  
MTpyError\_parameter\_number, 385  
MTpyError\_processing, 385  
MTpyError\_PT, 385

    MTpyError\_Tipper, 385  
    MTpyError\_ts\_data, 385  
    MTpyError\_value, 385  
    MTpyError\_Z, 385  
    MTStations (*class in mtpy.core*), 148  
    MTStations (*class in mtpy.core.mt\_stations*), 141  
    MTTimeError, 385  
    multiplymatrices\_incl\_errors() (*in module mtpy.utils.calculator*), 378

## N

n\_data (*mtpy.modeling.occam2d.data.Occam2DData property*), 261  
n\_data (*mtpy.modeling.occam2d.Occam2DData property*), 277  
n\_frequencies (*mtpy.modeling.occam2d.data.Occam2DData property*), 261  
n\_frequencies (*mtpy.modeling.occam2d.Occam2DData property*), 277  
n\_periods (*mtpy.core.transfer\_function.base.TFBase property*), 92  
n\_periods (*mtpy.modeling.modem.Data property*), 226  
n\_periods (*mtpy.modeling.modem.data.Data property*), 210  
n\_station\_x\_cells (*mtpy.modeling.simpeg.make\_2d\_mesh.StructuredMesh property*), 291  
n\_stations (*mtpy.core.mt\_data.MTData property*), 130  
n\_stations (*mtpy.modeling.occam2d.data.Occam2DData property*), 261  
n\_stations (*mtpy.modeling.occam2d.Occam2DData property*), 277  
n\_stations (*mtpy.MTData property*), 413  
n\_x\_padding (*mtpy.modeling.simpeg.make\_2d\_mesh.StructuredMesh property*), 291  
nearest\_index() (*in module mtpy.utils.calculator*), 378  
nodes\_east (*mtpy.modeling.modem.Model property*), 231  
nodes\_east (*mtpy.modeling.modem.model.Model property*), 215  
nodes\_east (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D property*), 314  
nodes\_east (*mtpy.modeling.StructuredGrid3D property*), 334  
nodes\_north (*mtpy.modeling.modem.Model property*), 231  
nodes\_north (*mtpy.modeling.modem.model.Model property*), 215  
nodes\_north (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D property*), 314  
nodes\_north (*mtpy.modeling.StructuredGrid3D property*), 334  
nodes\_z (*mtpy.modeling.modem.Model property*), 231

**n**

- nodes\_z (*mtpy.modeling.modem.model.Model* property), [Occam1DStartup](#) (*class* in *mtpy.modeling.occam1d*), [215](#)
- nodes\_z (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D* property), [Occam1DStartup](#) (*class* in *mtpy.modeling.occam1d.startup*), [247](#)
- nodes\_z (*mtpy.modeling.StructuredGrid3D* property), [Occam2DData](#) (*class* in *mtpy.modeling.occam2d*), [276](#)
- nonzero\_items (*mtpy.core.mt\_dataframe.MTDataFrame* property), [Occam2DData](#) (*class* in *mtpy.modeling.occam2d.data*), [260](#)
- nonzero\_items (*mtpy.core.MTDataFrame* property), [Occam2DModel](#) (*class* in *mtpy.modeling.occam2d*), [278](#)
- normalize\_L2() (*in module mtpy.processing.tf*), [260](#)
- normalizing\_imag (*mtpy.core.transfer\_function.z\_analysis.ZInvariant* property), [261](#)
- normalizing\_imag (*mtpy.core.transfer\_function.z\_analysis.zinvariants.ZInvariants* property), [89](#)
- normalizing\_real (*mtpy.core.transfer\_function.z\_analysis.ZInvariant* property), [261](#)
- normalizing\_real (*mtpy.core.transfer\_function.z\_analysis.zinvariants.ZInvariants* property), [88](#)
- north (*mtpy.core.mt\_dataframe.MTDataFrame* property), [138](#)
- north (*mtpy.core.mt\_location.MTLocation* property), [140](#)
- north (*mtpy.core.MTDataFrame* property), [146](#)
- north (*mtpy.core.MTLocation* property), [147](#)
- north (*mtpy.modeling.modem.station.Stations* property), [222](#)
- nout (*mtpy.processing.birrp.ScriptFile* property), [346](#)
- npes (*mtpy.processing.birrp.ScriptFile* property), [346](#)
- nref (*mtpy.processing.birrp.ScriptFile* property), [346](#)
- num\_sample\_rates (*mtpy.processing.kernel\_dataset.KernelDataset* property), [354](#)
- num\_sample\_rates (*mtpy.processing.KernelDataset* property), [374](#)
- number\_of\_active\_cells (*mtpy.modeling.simpeg.make\_2d\_mesh.QuadTreeMesh* property), [290](#)
- number\_of\_active\_cells (*mtpy.modeling.simpeg.make\_2d\_mesh.StructuredMesh* property), [291](#)
- nx (*mtpy.modeling.simpeg.make\_2d\_mesh.QuadTreeMesh* property), [290](#)
- nz (*mtpy.modeling.simpeg.make\_2d\_mesh.QuadTreeMesh* property), [290](#)

**O**

- Occam1DData (*class* in *mtpy.modeling.occam1d*), [249](#)
- Occam1DData (*class* in *mtpy.modeling.occam1d.data*), [237](#)
- Occam1DModel (*class* in *mtpy.modeling.occam1d*), [252](#)
- Occam1DModel (*class* in *mtpy.modeling.occam1d.model*), [240](#)
- Occam1DRun (*class* in *mtpy.modeling.occam1d*), [254](#)
- Occam1DRun (*class* in *mtpy.modeling.occam1d.run*), [247](#)

**P**

- padzeros() (*in module mtpy.processing.tf*), [360](#)
- period (*mtpy.core.mt\_dataframe.MTDataFrame* property), [138](#)
- period (*mtpy.core.MTDataFrame* property), [146](#)
- period (*mtpy.core.transfer\_function.base.TFBase* property), [92](#)
- period (*mtpy.imaging.plot\_mt\_response.PlotMTResponse* property), [182](#)
- period (*mtpy.imaging.PlotMTResponse* property), [196](#)
- period (*mtpy.modeling.modem.Data* property), [226](#)
- period (*mtpy.modeling.modem.data.Data* property), [210](#)
- period (*mtpy.modeling.ws3dinv.data.WSData* property), [294](#)

period (*mtpy.modeling.ws3dinv.WSData* property), 300  
period\_label\_dict (*mtpy.imaging.mtplot\_tools.plot\_settings.PlotSettings* property), 169  
period\_label\_dict (*mtpy.imaging.mtplot\_tools.PlotSettings* property), 177  
periods (*mtpy.modeling.simpeg.recipes.inversion\_1d.Simpeg1D* property), 285  
periods (*mtpy.modeling.simpeg.recipes.Simpeg1D* property), 287  
phase (*mtpy.core.Tipper* property), 155  
phase (*mtpy.core.transfer\_function.Tipper* property), 105  
phase (*mtpy.core.transfer\_function.tipper.Tipper* property), 96  
phase (*mtpy.core.transfer\_function.Z* property), 107  
phase (*mtpy.core.transfer\_function.z.Z* property), 98  
phase (*mtpy.core.Z* property), 157  
phase\_det (*mtpy.core.transfer\_function.Z* property), 107  
phase\_det (*mtpy.core.transfer\_function.z.Z* property), 98  
phase\_det (*mtpy.core.Z* property), 157  
phase\_distortion (*mtpy.core.transfer\_function.z\_analysis.ZInvariant* property), 89  
phase\_distortion (*mtpy.core.transfer\_function.z\_analysis* property), 88  
phase\_error (*mtpy.core.Tipper* property), 155  
phase\_error (*mtpy.core.transfer\_function.Tipper* property), 105  
phase\_error (*mtpy.core.transfer\_function.tipper.Tipper* property), 96  
phase\_error (*mtpy.core.transfer\_function.Z* property), 107  
phase\_error (*mtpy.core.transfer\_function.z.Z* property), 98  
phase\_error (*mtpy.core.Z* property), 157  
phase\_error\_det (*mtpy.core.transfer\_function.Z* property), 107  
phase\_error\_det (*mtpy.core.transfer\_function.z.Z* property), 98  
phase\_error\_det (*mtpy.core.Z* property), 157  
phase\_error\_xx (*mtpy.core.transfer\_function.Z* property), 107  
phase\_error\_xx (*mtpy.core.transfer\_function.z.Z* property), 98  
phase\_error\_xx (*mtpy.core.Z* property), 157  
phase\_error\_xy (*mtpy.core.transfer\_function.Z* property), 107  
phase\_error\_xy (*mtpy.core.transfer\_function.z.Z* property), 98  
phase\_error\_xy (*mtpy.core.Z* property), 157  
phase\_error\_yx (*mtpy.core.transfer\_function.Z* property), 107  
phase\_error\_yx (*mtpy.core.transfer\_function.z.Z* property), 99  
phase\_error\_yx (*mtpy.core.Z* property), 157  
phase\_error\_yy (*mtpy.core.transfer\_function.Z* property), 107  
phase\_error\_yy (*mtpy.core.transfer\_function.z.Z* property), 99  
phase\_error\_yy (*mtpy.core.Z* property), 157  
phase\_model\_error (*mtpy.core.Tipper* property), 155  
phase\_model\_error (*mtpy.core.transfer\_function.Tipper* property), 105  
phase\_model\_error (*mtpy.core.transfer\_function.tipper.Tipper* property), 97  
phase\_model\_error (*mtpy.core.transfer\_function.Z* property), 107  
phase\_model\_error (*mtpy.core.transfer\_function.z.Z* property), 99  
phase\_model\_error (*mtpy.core.Z* property), 157  
phase\_model\_error\_det  
    (*mtpy.core.transfer\_function.Z* property), 107  
phase\_model\_error\_det  
    (*mtpy.core.transfer\_function.z.Z* property), 99  
phase\_model\_error\_det (*mtpy.core.Z* property), 157  
phase\_model\_error\_xx (*mtpy.core.transfer\_function.Z* property), 107  
phase\_model\_error\_xx (*mtpy.core.transfer\_function.z.Z* property), 99  
phase\_model\_error\_xx (*mtpy.core.Z* property), 157  
phase\_model\_error\_xy (*mtpy.core.transfer\_function.Z* property), 107  
phase\_model\_error\_xy  
    (*mtpy.core.transfer\_function.z.Z* property), 99  
phase\_model\_error\_xy (*mtpy.core.Z* property), 157  
phase\_model\_error\_yx (*mtpy.core.transfer\_function.Z* property), 107  
phase\_model\_error\_yx (*mtpy.core.transfer\_function.z.Z* property), 99  
phase\_model\_error\_yx (*mtpy.core.Z* property), 157  
phase\_model\_error\_yy (*mtpy.core.transfer\_function.Z* property), 107  
phase\_model\_error\_yy (*mtpy.core.transfer\_function.z.Z* property), 99  
phase\_model\_error\_yy (*mtpy.core.Z* property), 157  
phase\_tensor (*mtpy.core.mt\_dataframe.MTDataFrame* property), 138  
phase\_tensor (*mtpy.core.MTDataFrame* property), 146  
phase\_tensor (*mtpy.core.transfer\_function.Z* property), 107  
phase\_tensor (*mtpy.core.transfer\_function.z.Z* property), 99  
phase\_tensor (*mtpy.core.Z* property), 157

**phase\_xx** (*mtpy.core.transfer\_function.Z* property), 108  
**phase\_xx** (*mtpy.core.transfer\_function.z.Z* property), 99  
**phase\_xx** (*mtpy.core.Z* property), 157  
**phase\_xy** (*mtpy.core.transfer\_function.Z* property), 108  
**phase\_xy** (*mtpy.core.transfer\_function.z.Z* property), 99  
**phase\_xy** (*mtpy.core.Z* property), 158  
**phase\_yx** (*mtpy.core.transfer\_function.Z* property), 108  
**phase\_yx** (*mtpy.core.transfer\_function.z.Z* property), 99  
**phase\_yx** (*mtpy.core.Z* property), 158  
**phase\_yy** (*mtpy.core.transfer\_function.Z* property), 108  
**phase\_yy** (*mtpy.core.transfer\_function.z.Z* property), 99  
**phase\_yy** (*mtpy.core.Z* property), 158  
**PhaseTensor** (class in *mtpy.core*), 152  
**PhaseTensor** (class in *mtpy.core.transfer\_function*), 102  
**PhaseTensor** (class in *mtpy.core.transfer\_function.pt*), 93  
**phimax** (*mtpy.core.PhaseTensor* property), 153  
**phimax** (*mtpy.core.transfer\_function.PhaseTensor* property), 103  
**phimax** (*mtpy.core.transfer\_function.pt.PhaseTensor* property), 95  
**phimax\_error** (*mtpy.core.PhaseTensor* property), 153  
**phimax\_error** (*mtpy.core.transfer\_function.PhaseTensor* property), 103  
**phimax\_error** (*mtpy.core.transfer\_function.pt.PhaseTensor* property), 95  
**phimax\_model\_error** (*mtpy.core.PhaseTensor* property), 154  
**phimax\_model\_error** (*mtpy.core.transfer\_function.PhaseTensor* property), 104  
**phimax\_model\_error** (*mtpy.core.transfer\_function.pt.PhaseTensor* property), 95  
**phimin** (*mtpy.core.PhaseTensor* property), 154  
**phimin** (*mtpy.core.transfer\_function.PhaseTensor* property), 104  
**phimin** (*mtpy.core.transfer\_function.pt.PhaseTensor* property), 95  
**phimin\_error** (*mtpy.core.PhaseTensor* property), 154  
**phimin\_error** (*mtpy.core.transfer\_function.PhaseTensor* property), 104  
**phimin\_error** (*mtpy.core.transfer\_function.pt.PhaseTensor* property), 95  
**phimin\_model\_error** (*mtpy.core.PhaseTensor* property), 154  
**phimin\_model\_error** (*mtpy.core.transfer\_function.PhaseTensor* property), 104  
**phimin\_model\_error** (*mtpy.core.transfer\_function.pt.PhaseTensor* property), 95  
**plot()** (in module *mtpy.utils.plot\_rms\_iterations*), 391  
**plot()** (*mtpy.imaging.mtplot\_tools.base*.*PlotBase* method), 163  
**plot()** (*mtpy.imaging.mtplot\_tools*.*PlotBase* method), 175  
**plot()** (*mtpy.imaging.plot\_mt\_response*.*PlotMTResponse* method), 182  
**plot()** (*mtpy.imaging.plot\_mt\_responses*.*PlotMultipleResponses* method), 183  
**plot()** (*mtpy.imaging.plot\_penetration\_depth\_1d*.*PlotPenetrationDepth1D* method), 183  
**plot()** (*mtpy.imaging.plot\_penetration\_depth\_map*.*PlotPenetrationDepthMap* method), 184  
**plot()** (*mtpy.imaging.plot\_phase\_tensor\_maps*.*PlotPhaseTensorMaps* method), 184  
**plot()** (*mtpy.imaging.plot\_phase\_tensor\_pseudosection*.*PlotPhaseTensorPseudoSection* method), 185  
**plot()** (*mtpy.imaging.plot\_pseudosection*.*PlotResPhasePseudoSection* method), 186  
**plot()** (*mtpy.imaging.plot\_pt*.*PlotPhaseTensor* method), 186  
**plot()** (*mtpy.imaging.plot\_residual\_pt\_maps*.*PlotResidualPTMaps* method), 188  
**plot()** (*mtpy.imaging.plot\_residual\_pt\_ps*.*PlotResidualPTPseudoSection* method), 190  
**plot()** (*mtpy.imaging.plot\_resphase\_maps*.*PlotResPhaseMaps* method), 191  
**plot()** (*mtpy.imaging.plot\_spectrogram*.*PlotTF* method), 192  
**plot()** (*mtpy.imaging.plot\_stations*.*PlotStations* method), 193  
**plot()** (*mtpy.imaging.plot\_strike*.*PlotStrike* method), 195  
**plot()** (*mtpy.imaging*.*PlotMTResponse* method), 196  
**plot()** (*mtpy.imaging*.*PlotMultipleResponses* method), 197  
**plot()** (*mtpy.imaging*.*PlotPenetrationDepth1D* method), 197  
**plot()** (*mtpy.imaging*.*PlotPenetrationDepthMap* method), 197  
**plot()** (*mtpy.imaging*.*PlotPhaseTensor* method), 197  
**plot()** (*mtpy.imaging*.*PlotPhaseTensorMaps* method), 198  
**plot()** (*mtpy.imaging*.*PlotPhaseTensorPseudoSection* method), 199  
**plot()** (*mtpy.imaging*.*PlotResidualPTMaps* method), 201  
**plot()** (*mtpy.imaging*.*PlotResidualPTPseudoSection* method), 203  
**plot()** (*mtpy.imaging*.*PlotResPhaseMaps* method), 200  
**plot()** (*mtpy.imaging*.*PlotResPhasePseudoSection* method), 200  
**plot()** (*mtpy.imaging*.*PlotStations* method), 203  
**plot()** (*mtpy.imaging*.*PlotStrike* method), 205  
**plot()** (*mtpy.modeling.occam1d*.*Plot1DResponse* method), 258  
**plot()** (*mtpy.modeling.occam1d*.*plot\_l2*.*PlotOccam1DL2* method), 243  
**plot()** (*mtpy.modeling.occam1d*.*plot\_response*.*Plot1DResponse* method), 245

plot() (*mtpy.modeling.occam1d.PlotOccam1DL2 method*), 260  
plot() (*mtpy.modeling.plots.plot\_mesh.PlotMesh method*), 283  
plot() (*mtpy.modeling.plots.plot\_modem\_rms.PlotRMS method*), 283  
plot() (*mtpy.modeling.plots.PlotMesh method*), 284  
plot() (*mtpy.modeling.plots.PlotRMS method*), 284  
plot() (*mtpy.modeling.winglink.PlotMisfitPseudoSection method*), 320  
plot() (*mtpy.modeling.winglink.PlotPseudoSection method*), 323  
plot() (*mtpy.modeling.winglink.PlotResponse method*), 325  
Plot1DResponse (class in *mtpy.modeling.occam1d*), 256  
Plot1DResponse (class in *mtpy.modeling.occam1d.plot\_response*), 244  
plot\_data() (in module *mtpy.utils.basemap\_tools*), 377  
plot\_depth\_of\_penetration() (*mtpy.core.mt.MT method*), 114  
plot\_depth\_of\_penetration() (*mtpy.MT method*), 397  
plot\_east (*mtpy.modeling.modem.Model property*), 231  
plot\_east (*mtpy.modeling.modem.model.Model property*), 215  
plot\_east (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D property*), 314  
plot\_east (*mtpy.modeling.StructuredGrid3D property*), 334  
plot\_errorbar() (in module *mtpy.imaging.mtplot\_tools*), 178  
plot\_errorbar() (in module *mtpy.imaging.mtplot\_tools.plotters*), 170  
plot\_iteration() (*mtpy.modeling.simpeg.recipes.Simpeg2D method*), 288  
plot\_mesh() (*mtpy.modeling.modem.Model method*), 231  
plot\_mesh() (*mtpy.modeling.modem.model.Model method*), 215  
plot\_mesh() (*mtpy.modeling.occam2d.Mesh method*), 275  
plot\_mesh() (*mtpy.modeling.occam2d.mesh.Mesh method*), 265  
plot\_mesh() (*mtpy.modeling.simpeg.make\_2d\_mesh.QuadTreeMesh method*), 403  
plot\_mesh() (*mtpy.modeling.simpeg.make\_2d\_mesh.StructuredMesh method*), 414  
plot\_mesh() (*mtpy.modeling.simpeg.make\_2d\_mesh.StructuredMesh method*), 291  
plot\_mesh() (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D method*), 314  
plot\_mesh() (*mtpy.modeling.StructuredGrid3D method*), 334  
plot\_model\_error (*mtpy.imaging.plot\_mt\_response.PlotMultipleResponses property*), 182  
plot\_model\_error (*mtpy.imaging.plot\_mt\_responses.PlotMultipleResponses property*), 183  
plot\_model\_error (*mtpy.imaging.PlotMTResponse property*), 196  
plot\_model\_error (*mtpy.imaging.PlotMultipleResponses property*), 197  
plot\_model\_fitting() (*mtpy.modeling.simpeg.recipes.inversion\_1d.Simpeg1D method*), 285  
plot\_model\_fitting() (*mtpy.modeling.simpeg.recipes.Simpeg1D method*), 287  
plot\_mt\_response() (*mtpy.core.mt.MT method*), 114  
plot\_mt\_response() (*mtpy.core.mt\_collection.MTCollection method*), 120  
plot\_mt\_response() (*mtpy.core.mt\_data.MTData method*), 130  
plot\_mt\_response() (*mtpy.MT method*), 397  
plot\_mt\_response() (*mtpy.MTCollection method*), 402  
plot\_mt\_response() (*mtpy.MTData method*), 413  
plot\_north (*mtpy.modeling.modem.Model property*), 232  
plot\_north (*mtpy.modeling.modem.model.Model property*), 216  
plot\_north (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D property*), 314  
plot\_north (*mtpy.modeling.StructuredGrid3D property*), 334  
plot\_penetration\_depth\_1d() (*mtpy.core.mt\_collection.MTCollection method*), 120  
plot\_penetration\_depth\_1d() (*mtpy.core.mt\_data.MTData method*), 131  
plot\_penetration\_depth\_1d() (*mtpy.MTCollection method*), 403  
plot\_penetration\_depth\_1d() (*mtpy.MTData method*), 413  
plot\_penetration\_depth\_map() (*mtpy.core.mt\_collection.MTCollection method*), 121  
plot\_penetration\_depth\_map() (*mtpy.core.mt\_data.MTData method*), 131  
plot\_penetration\_depth\_map() (*mtpy.MTCollection method*), 403  
plot\_penetration\_depth\_map() (*mtpy.MTData method*), 414  
plot\_phase() (in module *mtpy.imaging.mtplot\_tools*), 179  
plot\_phase() (in module *mtpy.imaging.mtplot\_tools.plotters*), 171  
plot\_phase\_tensor() (*mtpy.core.mt.MT method*), 114  
plot\_phase\_tensor() (*mtpy.core.mt\_data.MTData method*), 131

`(mtpy.core.mt_collection.MTCollection  
method), 121`  
`plot_phase_tensor() (mtpy.core.mt_data.MTData  
method), 132`  
`plot_phase_tensor() (mtpy.MT method), 397`  
`plot_phase_tensor() (mtpy.MTCollection method),  
404`  
`plot_phase_tensor() (mtpy.MTData method), 414`  
`plot_phase_tensor_map()  
(mtpy.core.mt_collection.MTCollection  
method), 121`  
`plot_phase_tensor_map()  
(mtpy.core.mt_data.MTData method), 132`  
`plot_phase_tensor_map() (mtpy.MTCollection  
method), 404`  
`plot_phase_tensor_map() (mtpy.MTData method),  
414`  
`plot_phase_tensor_pseudosection()  
(mtpy.core.mt_collection.MTCollection  
method), 122`  
`plot_phase_tensor_pseudosection()  
(mtpy.core.mt_data.MTData method), 132`  
`plot_phase_tensor_pseudosection()  
(mtpy.MTCollection method), 404`  
`plot_phase_tensor_pseudosection()  
(mtpy.MTData method), 415`  
`plot_pt_lateral() (in module  
mtpy.imaging.mtplot_tools), 179`  
`plot_pt_lateral() (in module  
mtpy.imaging.mtplot_tools.plotters), 171`  
`plot_residual_phase_tensor()  
(mtpy.core.mt_collection.MTCollection  
method), 122`  
`plot_residual_phase_tensor() (mtpy.MTCollection  
method), 404`  
`plot_residual_phase_tensor_maps()  
(mtpy.core.mt_data.MTData method), 133`  
`plot_residual_phase_tensor_maps()  
(mtpy.MTData method), 415`  
`plot_resistivity() (in module  
mtpy.imaging.mtplot_tools), 180`  
`plot_resistivity() (in module  
mtpy.imaging.mtplot_tools.plotters), 171`  
`plot_resistivity_phase_maps()  
(mtpy.core.mt_collection.MTCollection  
method), 122`  
`plot_resistivity_phase_maps()  
(mtpy.core.mt_data.MTData method), 133`  
`plot_resistivity_phase_maps()  
(mtpy.MTCollection method), 405`  
`plot_resistivity_phase_maps() (mtpy.MTData  
method), 415`  
`plot_resistivity_phase_pseudosections()  
(mtpy.core.mt_collection.MTCollection  
method), 123`  
`plot_resistivity_phase_pseudosections()  
(mtpy.core.mt_data.MTData method), 133`  
`plot_resistivity_phase_pseudosections()  
(mtpy.MTCollection method), 405`  
`plot_resistivity_phase_pseudosections()  
(mtpy.MTData method), 415`  
`plot_response() (mtpy.modeling.simpeg.recipes.inversion_1d.Simpeg1D  
method), 285`  
`plot_response() (mtpy.modeling.simpeg.recipes.Simpeg1D  
method), 287`  
`plot_responses() (mtpy.modeling.simpeg.recipes.Simpeg2D  
method), 288`  
`plot_rms() (mtpy.modeling.modem.Residual method),  
236`  
`plot_rms() (mtpy.modeling.modem.residual.Residual  
method), 220`  
`plot_rms_per_period()  
(mtpy.modeling.modem.Residual method),  
237`  
`plot_rms_per_period()  
(mtpy.modeling.modem.residual.Residual  
method), 221`  
`plot_stations() (mtpy.core.mt_collection.MTCollection  
method), 123`  
`plot_stations() (mtpy.core.mt_data.MTData  
method), 133`  
`plot_stations() (mtpy.MTCollection method), 405`  
`plot_stations() (mtpy.MTData method), 416`  
`plot_strike() (mtpy.core.mt_collection.MTCollection  
method), 123`  
`plot_strike() (mtpy.core.mt_data.MTData method),  
134`  
`plot_strike() (mtpy.MTCollection method), 406`  
`plot_strike() (mtpy.MTData method), 416`  
`plot_tikhonov_curve()  
(mtpy.modeling.simpeg.recipes.Simpeg2D  
method), 289`  
`plot_tipper_lateral() (in module  
mtpy.imaging.mtplot_tools), 180`  
`plot_tipper_lateral() (in module  
mtpy.imaging.mtplot_tools.plotters), 172`  
`plot_tipper_map() (mtpy.core.mt_data.MTData  
method), 134`  
`plot_tipper_map() (mtpy.MTData method), 416`  
`plot_z (mtpy.modeling.modem.Model property), 232`  
`plot_z (mtpy.modeling.modem.model.Model property),  
216`  
`plot_z (mtpy.modeling.structured_mesh_3d.StructuredGrid3D  
property), 314`  
`plot_z (mtpy.modeling.StructuredGrid3D property), 334`  
`PlotBase (class in mtpy.imaging.mtplot_tools), 175`  
`PlotBase (class in mtpy.imaging.mtplot_tools.base), 163`  
`PlotBaseMaps (class in mtpy.imaging.mtplot_tools), 176`

PlotBaseMaps	(class in mtpy.imaging.mtplot_tools.base),	164
PlotBaseProfile	(class in mtpy.imaging.mtplot_tools),	176
PlotBaseProfile	(class in mtpy.imaging.mtplot_tools.base),	164
PlotMesh	(class in mtpy.modeling.plots),	284
PlotMesh	(class in mtpy.modeling.plots.plot_mesh),	283
PlotMisfitPseudoSection	(class in mtpy.modeling.winglink),	319
PlotMTResponse	(class in mtpy.imaging),	196
PlotMTResponse	(class in mtpy.imaging.plot_mt_response),	182
PlotMultipleResponses	(class in mtpy.imaging),	196
PlotMultipleResponses	(class in mtpy.imaging.plot_mt_responses),	182
PlotOccam1DL2	(class in mtpy.modeling.occam1d),	259
PlotOccam1DL2	(class in mtpy.modeling.occam1d.plot_l2),	243
PlotPenetrationDepth1D	(class in mtpy.imaging),	197
PlotPenetrationDepth1D	(class in mtpy.imaging.plot_penetration_depth_1d),	183
PlotPenetrationDepthMap	(class in mtpy.imaging),	197
PlotPenetrationDepthMap	(class in mtpy.imaging.plot_penetration_depth_map),	183
PlotPhaseTensor	(class in mtpy.imaging),	197
PlotPhaseTensor	(class in mtpy.imaging.plot_pt),	186
PlotPhaseTensorMaps	(class in mtpy.imaging),	197
PlotPhaseTensorMaps	(class in mtpy.imaging.plot_phase_tensor_maps),	184
PlotPhaseTensorPseudoSection	(class in mtpy.imaging),	198
PlotPhaseTensorPseudoSection	(class in mtpy.imaging.plot_phase_tensor_pseudosection),	185
PlotPseudoSection	(class in mtpy.modeling.winglink),	321
PlotResidualPTMaps	(class in mtpy.imaging),	200
PlotResidualPTMaps	(class in mtpy.imaging.plot_residual_pt_maps),	186
PlotResidualPTPseudoSection	(class in mtpy.imaging),	202
PlotResidualPTPseudoSection	(class in mtpy.imaging.plot_residual_pt_ps),	188
PlotResPhaseMaps	(class in mtpy.imaging),	199
PlotResPhaseMaps	(class in mtpy.imaging.plot_resphase_maps),	191
PlotResPhasePseudoSection	(class in mtpy.imaging),	200
PlotResPhasePseudoSection	(class in mtpy.imaging.plot_resphase_pseudo_section),	186
PlotResponse	(class in mtpy.modeling.winglink),	324
plotResponses()	(in module mtpy.modeling.winglinktools),	326
PlotRMS	(class in mtpy.modeling.plots),	284
PlotRMS	(class in mtpy.modeling.plots.plot_modem_rms),	283
PlotSettings	(class in mtpy.imaging.mtplot_tools),	176
PlotSettings	(class in mtpy.imaging.mtplot_tools.plot_settings),	169
PlotStations	(class in mtpy.imaging),	203
PlotStations	(class in mtpy.imaging.plot_stations),	193
PlotStrike	(class in mtpy.imaging),	203
PlotStrike	(class in mtpy.imaging.plot_strike),	193
PlotTF	(class in mtpy.imaging.plot_spectrogram),	192
print_mini_summary	(mtpy.processing.run_summary.RunSummary property),	357
print_mini_summary	(mtpy.processing.RunSummary property),	376
print_suspect_stations()	(mtpy.modeling.plots.plot_modem_rms.PlotRMS method),	283
print_suspect_stations()	(mtpy.modeling.plots.PlotRMS method),	284
process()	(mtpy.processing.aurora.process_aurora.AuroraProcessing method),	340
process()	(mtpy.processing.AuroraProcessing method),	370
process_single_sample_rate()	(mtpy.processing.aurora.process_aurora.AuroraProcessing method),	340
process_single_sample_rate()	(mtpy.processing.AuroraProcessing method),	371
processing_id	(mtpy.processing.kernel_datasetKernelDataset property),	354
processing_id	(mtpy.processing.KernelDataset prop- erty),	375
profile_offset	(mtpy.core.mt_dataframe.MTDataFrame property),	138
profile_offset	(mtpy.core.MTDataFrame property),	146
project_onto_profile_line()	(mtpy.core.mt_location.MTLocation method),	140
project_onto_profile_line()	(mtpy.core.MTLocation method),	147
project_point()	(in module mtpy.utils.gis_tools),	390
project_point_ll2utm()	(in module mtpy.utils.gis_tools),	390
project_point_utm2ll()	(in module	

*mtpy.utils.gis\_tools), 390*  
**project\_stations\_on\_topography()** (*mtpy.core.mt\_stations.MTStations method*), 142  
**project\_stations\_on\_topography()** (*mtpy.core.MTStations method*), 150  
**propagate\_error\_polar2rect()** (*in module mtpy.utils.calculator*), 379  
**propagate\_error\_rect2polar()** (*in module mtpy.utils.calculator*), 379  
**pt** (*mtpy.core.MT property*), 115  
**pt** (*mtpy.core.PhaseTensor property*), 154  
**pt** (*mtpy.core.transfer\_function.PhaseTensor property*), 104  
**pt** (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 95  
**pt** (*MTP property*), 397  
**pt\_error** (*mtpy.core.PhaseTensor property*), 154  
**pt\_error** (*mtpy.core.transfer\_function.PhaseTensor property*), 104  
**pt\_error** (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 95  
**pt\_model\_error** (*mtpy.core.PhaseTensor property*), 154  
**pt\_model\_error** (*mtpy.core.transfer\_function.PhaseTensor property*), 104  
**pt\_model\_error** (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 95

**Q**

**QuadTreeMesh** (*class in mtpy.modeling.simpeg.make\_2d\_mesh*), 290

**R**

**read()** (*in module mtpy.utils.plot\_rms\_iterations*), 391  
**read1columntext()** (*in module mtpy.utils.filehandling*), 387  
**read\_2c2\_file()** (*in module mtpy.utils.filehandling*), 387  
**read\_birrp\_config\_fn()** (*mtpy.processing.birrp.J2Edi method*), 342  
**read\_config\_file()** (*mtpy.processing.birrp.BIRRPPParameters method*), 341  
**read\_configfile()** (*in module mtpy.utils.configfile*), 381  
**read\_control\_file()** (*mtpy.modeling.modem.control\_fwd.ControlFwd method*), 206  
**read\_control\_file()** (*mtpy.modeling.modem.control\_inv.ControlInv method*), 208  
**read\_control\_file()** (*mtpy.modeling.modem.ControlFwd method*), 223  
**read\_control\_file()** (*mtpy.modeling.modem.ControlInv method*), 224  
**read\_cov\_file()** (*mtpy.modeling.modem.convariance.Covariance method*), 208  
**read\_cov\_file()** (*mtpy.modeling.modem.Covariance method*), 225  
**read\_data\_file()** (*mtpy.modeling.modem.Data method*), 226  
**read\_data\_file()** (*mtpy.modeling.modem.data.Data method*), 210  
**read\_data\_file()** (*mtpy.modeling.occam1d.data.Occam1DData method*), 238  
**read\_data\_file()** (*mtpy.modeling.occam1d.Occam1DData method*), 250  
**read\_data\_file()** (*mtpy.modeling.occam2d.data.Occam2DData method*), 261  
**read\_data\_file()** (*mtpy.modeling.occam2d.Occam2DData method*), 277  
**read\_data\_file()** (*mtpy.modeling.ws3dinv.data.WSData method*), 294  
**read\_data\_file()** (*mtpy.modeling.ws3dinv.WSData method*), 300  
**read\_data\_header()** (*in module mtpy.utils.filehandling*), 387  
**read\_geotiff()** (*in module mtpy.utils.filehandling*), 387  
**read\_gocad\_sgrid\_file()** (*mtpy.modeling.modem.Model method*), 232  
**read\_gocad\_sgrid\_file()** (*mtpy.modeling.modem.model.Model method*), 216  
**read\_iter\_file()** (*mtpy.modeling.occam1d.model.Occam1DModel method*), 241  
**read\_iter\_file()** (*mtpy.modeling.occam1d.Occam1DModel method*), 253  
**read\_iter\_file()** (*mtpy.modeling.occam2d.model.Occam2DModel method*), 267  
**read\_iter\_file()** (*mtpy.modeling.occam2d.Occam2DModel method*), 279  
**read\_mesh\_file()** (*mtpy.modeling.occam2d.Mesh method*), 275  
**read\_mesh\_file()** (*mtpy.modeling.occam2d.mesh.Mesh method*), 265  
**read\_model\_file()** (*in module mtpy.modeling.winglink*), 326  
**read\_model\_file()** (*mtpy.modeling.modem.Model method*), 232  
**read\_model\_file()** (*mtpy.modeling.modem.model.Model method*), 216  
**read\_model\_file()** (*mtpy.modeling.occam1d.model.Occam1DModel method*), 242  
**read\_model\_file()** (*mtpy.modeling.occam1d.Occam1DModel method*)

method), 253

read\_output\_file() (in module mtpy.modeling.winglink), 326

read\_pts() (mtpy.analysis.residual\_phase\_tensor.Residual method), 83

read\_regularization\_file() (mtpy.modeling.occam2d.Regularization method), 281

read\_regularization\_file() (mtpy.modeling.occam2d.regularization.Regularization method), 269

read\_residual\_file() (mtpy.modeling.modem.Residual method), 237

read\_residual\_file() (mtpy.modeling.modem.residual.Residual method), 221

read\_resp\_file() (mtpy.modeling.occam1d.data.Occam1D method), 239

read\_resp\_file() (mtpy.modeling.occam1d.Occam1DData method), 250

read\_sgrid\_file() (mtpy.modeling.gocad.Sgrid method), 305

read\_startup\_file() (mtpy.modeling.occam1d.Occam1DStartup method), 255

read\_startup\_file() (mtpy.modeling.occam1d.startup.Occam1DStartup method), 247

read\_startup\_file() (mtpy.modeling.ws3dinv.startup.WSStartup method), 295

read\_startup\_file() (mtpy.modeling.ws3dinv.WSStartup method), 301

read\_station\_file() (mtpy.modeling.ws3dinv.stations.WSStation method), 296

read\_station\_file() (mtpy.modeling.ws3dinv.WSStation method), 302

read\_stationsdatafile() (in module mtpy.utils.filehandling), 388

read\_surface\_ascii() (in module mtpy.utils.filehandling), 388

read\_survey\_config\_fn() (mtpy.processing.birrp.J2Edi method), 342

read\_survey\_configfile() (in module mtpy.utils.configfile), 381

read\_survey\_txt\_file() (in module mtpy.utils.configfile), 382

read\_ts\_file() (in module mtpy.utils.filehandling), 388

read\_ts\_header() (in module mtpy.utils.filehandling), 388

readModelFile() (in module mtpy.modeling.winglinktools), 327

readOutputFile() (in module mtpy.modeling.winglinktools), 327

reassigned\_smethod() (in module mtpy.processing.tf), 360

reassigned\_stft() (in module mtpy.processing.tf), 361

recursive\_glob() (in module mtpy.utils.edi\_folders), 383

redraw\_plot() (mtpy.imaging.mtplot\_tools.base.PlotBase method), 163

redraw\_plot() (mtpy.imaging.mtplot\_tools.PlotBase method), 175

redraw\_plot() (mtpy.imaging.plot\_spectrogram.PlotTF method), 192

redraw\_plot() (mtpy.modeling.occam1d.Plot1DResponse method), 258

redraw\_plot() (mtpy.modeling.occam1d.plot\_response.Plot1DResponse method), 245

redraw\_plot() (mtpy.modeling.winglink.PlotMisfitPseudoSection method), 320

redraw\_plot() (mtpy.modeling.winglink.PlotPseudoSection method), 323

redraw\_plot() (mtpy.modeling.winglink.PlotResponse method), 325

reference\_model (mtpy.modeling.simpeg.recipes.Simpeg2D property), 289

register\_cmaps() (in module mtpy.imaging.mtcOLORS), 181

Regularization (class in mtpy.modeling.occam2d), 279

Regularization (class in mtpy.modeling.occam2d.regularization), 268

regularization (mtpy.modeling.simpeg.recipes.Simpeg2D property), 289

rel\_east (mtpy.modeling.modem.station.Stations property), 222

rel\_elev (mtpy.modeling.modem.station.Stations property), 222

rel\_north (mtpy.modeling.modem.station.Stations property), 222

remote\_df (mtpy.processing.kernel\_dataset.KernelDataset property), 354

remote\_df (mtpy.processing.KernelDataset property), 375

remote\_mth5\_path (mtpy.processing.kernel\_dataset.KernelDataset property), 354

remote\_mth5\_path (mtpy.processing.KernelDataset property), 375

remote\_station\_id (mtpy.processing.kernel\_dataset.KernelDataset property), 354

**remote\_station\_id** (*mtpy.processing.KernelDataset property*), 375  
**remove\_component()** (*mtpy.core.mt.MT method*), 115  
**remove\_component()** (*mtpy.MT method*), 397  
**remove\_distortion()** (*mtpy.core.mt.MT method*), 115  
**remove\_distortion()** (*mtpy.core.transfer\_function.Z method*), 108  
**remove\_distortion()** (*mtpy.core.transfer\_function.z.Z method*), 99  
**remove\_distortion()** (*mtpy.core.Z method*), 158  
**remove\_distortion()** (*mtpy.MT method*), 398  
**remove\_distortion\_from\_z\_object()** (*in module mtpy.core.transfer\_function.z\_analysis*), 90  
**remove\_distortion\_from\_z\_object()** (*in module mtpy.core.transfer\_function.z\_analysis.distortion*), 85  
**remove\_periodic\_noise()** (*in module mtpy.processing.filter*), 348  
**remove\_ss()** (*mtpy.core.transfer\_function.Z method*), 108  
**remove\_ss()** (*mtpy.core.transfer\_function.z.Z method*), 100  
**remove\_ss()** (*mtpy.core.Z method*), 158  
**remove\_static\_shift()** (*mtpy.core.mt.MT method*), 116  
**remove\_static\_shift()** (*mtpy.MT method*), 398  
**remove\_station()** (*mtpy.core.mt\_data.MTData method*), 134  
**remove\_station()** (*mtpy.MTData method*), 416  
**reorient\_data2D()** (*in module mtpy.utils.calculator*), 379  
**reorient\_files()** (*in module mtpy.utils.filehandling*), 388  
**res\_det** (*mtpy.core.transfer\_function.Z property*), 109  
**res\_det** (*mtpy.core.transfer\_function.z.Z property*), 100  
**res\_det** (*mtpy.core.Z property*), 159  
**res\_error\_det** (*mtpy.core.transfer\_function.Z property*), 109  
**res\_error\_det** (*mtpy.core.transfer\_function.z.Z property*), 100  
**res\_error\_det** (*mtpy.core.Z property*), 159  
**res\_error\_xx** (*mtpy.core.transfer\_function.Z property*), 109  
**res\_error\_xx** (*mtpy.core.transfer\_function.z.Z property*), 100  
**res\_error\_xx** (*mtpy.core.Z property*), 159  
**res\_error\_xy** (*mtpy.core.transfer\_function.Z property*), 109  
**res\_error\_xy** (*mtpy.core.transfer\_function.z.Z property*), 100  
**res\_error\_xy** (*mtpy.core.Z property*), 159  
**res\_error\_yx** (*mtpy.core.transfer\_function.Z property*), 109  
**res\_error\_yx** (*mtpy.core.transfer\_function.z.Z property*), 100  
**res\_error\_yy** (*mtpy.core.Z property*), 159  
**res\_error\_yy** (*mtpy.core.transfer\_function.z.Z property*), 109  
**res\_error\_yy** (*mtpy.core.Z property*), 159  
**res\_model\_error\_det** (*mtpy.core.transfer\_function.Z property*), 109  
**res\_model\_error\_det** (*mtpy.core.transfer\_function.z.Z property*), 101  
**res\_model\_error\_det** (*mtpy.core.Z property*), 159  
**res\_model\_error\_xx** (*mtpy.core.transfer\_function.Z property*), 109  
**res\_model\_error\_xx** (*mtpy.core.transfer\_function.z.Z property*), 101  
**res\_model\_error\_xx** (*mtpy.core.Z property*), 159  
**res\_model\_error\_xy** (*mtpy.core.transfer\_function.Z property*), 109  
**res\_model\_error\_xy** (*mtpy.core.transfer\_function.z.Z property*), 101  
**res\_model\_error\_xy** (*mtpy.core.Z property*), 159  
**res\_model\_error\_yx** (*mtpy.core.transfer\_function.Z property*), 109  
**res\_model\_error\_yx** (*mtpy.core.transfer\_function.z.Z property*), 101  
**res\_model\_error\_yx** (*mtpy.core.Z property*), 159  
**res\_model\_error\_yy** (*mtpy.core.transfer\_function.Z property*), 109  
**res\_model\_error\_yy** (*mtpy.core.transfer\_function.z.Z property*), 101  
**res\_model\_error\_yy** (*mtpy.core.Z property*), 159  
**res\_xx** (*mtpy.core.transfer\_function.Z property*), 109  
**res\_xx** (*mtpy.core.transfer\_function.z.Z property*), 101  
**res\_xx** (*mtpy.core.Z property*), 159  
**res\_xy** (*mtpy.core.transfer\_function.Z property*), 109  
**res\_xy** (*mtpy.core.transfer\_function.z.Z property*), 101  
**res\_xy** (*mtpy.core.Z property*), 159  
**res\_yx** (*mtpy.core.transfer\_function.Z property*), 109  
**res\_yx** (*mtpy.core.transfer\_function.z.Z property*), 101  
**res\_yx** (*mtpy.core.Z property*), 159  
**res\_yy** (*mtpy.core.transfer\_function.Z property*), 110  
**res\_yy** (*mtpy.core.transfer\_function.z.Z property*), 101  
**res\_yy** (*mtpy.core.Z property*), 159  
**Residual** (*class in mtpy.modeling.modem*), 235  
**Residual** (*class in mtpy.modeling.modem.residual*), 219  
**ResidualPhaseTensor** (*class in mtpy.analysis.residual\_phase\_tensor*), 83  
**resistivity** (*mtpy.core.transfer\_function.Z property*), 110  
**resistivity** (*mtpy.core.transfer\_function.z.Z property*), 101  
**resistivity** (*mtpy.core.Z property*), 159  
**resistivity\_error** (*mtpy.core.transfer\_function.Z prop-*

property), 110  
resistivity\_error (*mtpy.core.transfer\_function.z.Z* property), 101  
resistivity\_error (*mtpy.core.Z* property), 159  
resistivity\_model\_error  
    (*mtpy.core.transfer\_function.Z* property), 110  
resistivity\_model\_error  
    (*mtpy.core.transfer\_function.z.Z* property), 101  
resistivity\_model\_error (*mtpy.core.Z* property), 160  
resize\_output() (*mtpy.modeling.errors.ModelErrors* method), 304  
restrict\_run\_intervals\_to\_simultaneous()  
    (*mtpy.processing.kernel\_dataset.KernelDataset* method), 354  
restrict\_run\_intervals\_to\_simultaneous()  
    (*mtpy.processing.KernelDataset* method), 375  
restrict\_to\_station\_list() (in module  
    *mtpy.processing.kernel\_dataset*), 356  
rhophi2z() (in module *mtpy.utils.calculator*), 379  
rms\_array (*mtpy.modeling.plots.plot\_modem\_rms.PlotRMS* property), 283  
rms\_array (*mtpy.modeling.plots.PlotRMS* property), 284  
rms\_cmap (*mtpy.modeling.plots.plot\_modem\_rms.PlotRMS* property), 284  
rms\_cmap (*mtpy.modeling.plots.PlotRMS* property), 284  
rms\_per\_period\_all (*mtpy.modeling.modem.Residual* property), 237  
rms\_per\_period\_all (*mtpy.modeling.modem.residual.Residual* property), 221  
rms\_per\_period\_all (*mtpy.modeling.plots.plot\_modem\_rms.PlotRMS* property), 284  
rms\_per\_period\_all (*mtpy.modeling.plots.PlotRMS* property), 284  
rms\_per\_period\_per\_component  
    (*mtpy.modeling.modem.Residual* property), 237  
rms\_per\_period\_per\_component  
    (*mtpy.modeling.modem.residual.Residual* property), 221  
rms\_per\_station (*mtpy.modeling.plots.plot\_modem\_rms.PlotRMS* property), 284  
rms\_per\_station (*mtpy.modeling.plots.PlotRMS* property), 284  
robust\_smethod() (in module *mtpy.processing.tf*), 362  
robust\_stft\_L() (in module *mtpy.processing.tf*), 363  
robust\_stft\_median() (in module  
    *mtpy.processing.tf*), 363  
robust\_wvd() (in module *mtpy.processing.tf*), 364  
rotate() (*mtpy.core.mt.MT* method), 116  
rotate() (*mtpy.core.mt\_data.MTData* method), 134  
rotate() (in *mtpy.core.transfer\_function.base.TFBase* method), 92  
rotate() (*mtpy.MT* method), 399  
rotate() (*mtpy.MTData* method), 417  
rotate\_matrix\_errors() (in module  
    *mtpy.utils.calculator*), 379  
rotate\_matrix\_with\_errors() (in module  
    *mtpy.utils.calculator*), 379  
rotate\_mesh() (in module *mtpy.modeling.mesh\_tools*), 307  
rotate\_stations() (*mtpy.core.mt\_stations.MTStations* method), 143  
rotate\_stations() (*mtpy.core.MTStations* method), 150  
rotate\_stations() (*mtpy.modeling.modem.station.Stations* method), 222  
rotate\_vector\_with\_errors() (in module  
    *mtpy.utils.calculator*), 380  
rotation\_angle (*mtpy.core.mt.MT* property), 116  
rotation\_angle (*mtpy.imaging.mtplot\_tools.base.PlotBaseProfile* property), 164  
rotation\_angle (*mtpy.imaging.mtplot\_tools.PlotBaseProfile* property), 176  
rotation\_angle (*mtpy.imaging.plot\_mt\_response.PlotMTResponse* property), 182  
rotation\_angle (*mtpy.imaging.plot\_mt\_responses.PlotMultipleResponses* property), 183  
rotation\_angle (*mtpy.imaging.plot\_phase\_tensor\_maps.PlotPhaseTensor* property), 184  
rotation\_angle (*mtpy.imaging.plot\_pt.PlotPhaseTensor* property), 186  
rotation\_angle (*mtpy.imaging.plot\_residual\_pt\_maps.PlotResidualPTM* property), 188  
rotation\_angle (*mtpy.imaging.plot\_residual\_pt\_ps.PlotResidualPTPseudoSection* property), 190  
rotation\_angle (*mtpy.imaging.plot\_strike.PlotStrike* property), 195  
rotation\_angle (*mtpy.imaging.PlotMTResponse* property), 196  
rotation\_angle (*mtpy.imaging.PlotMultipleResponses* property), 197  
rotation\_angle (*mtpy.imaging.PlotPhaseTensor* property), 197  
rotation\_angle (*mtpy.imaging.PlotPhaseTensorMaps* property), 198  
rotation\_angle (*mtpy.imaging.PlotResidualPTMaps* property), 201  
rotation\_angle (*mtpy.imaging.PlotResidualPTPseudoSection* property), 203  
rotation\_angle (*mtpy.imaging.PlotStrike* property), 205  
rotation\_angle (*mtpy.MT* property), 399  
round\_to\_step() (in module  
    *mtpy.imaging.mtplot\_tools.utils*), 173

`roundsfc() (in module mtpy.utils.calculator), 380`

`rrhx_metadata (mtpy.core.mt.MT property), 116`

`rrhx_metadata (mtpy.MT property), 399`

`rrhy_metadata (mtpy.core.mt.MT property), 116`

`rrhy_metadata (mtpy.MT property), 399`

`run() (in module mtpy.processing.birrp), 346`

`run_fixed_layer_inversion()`  
`(mtpy.modeling.simpeg.recipes.inversion_1d.Simpeg1D method), 286`

`run_fixed_layer_inversion()`  
`(mtpy.modeling.simpeg.recipes.Simpeg1D method), 287`

`run_inversion() (mtpy.modeling.simpeg.recipes.Simpeg2D method), 289`

`run_occam1d() (mtpy.modeling.occam1d.Occam1DRun method), 254`

`run_occam1d() (mtpy.modeling.occam1d.run.Occam1DRun method), 247`

`run_summary (mtpy.processing.base.BaseProcessing property), 341`

`RunSummary (class in mtpy.processing), 376`

`RunSummary (class in mtpy.processing.run_summary), 356`

**S**

`sample_rate (mtpy.processing.kernel_datasetKernelDataset property), 354`

`sample_rate (mtpy.processing.KernelDataset property), 375`

`save_dir (mtpy.gis.shapefile_creator.ShapefileCreator property), 162`

`save_figure() (mtpy.imaging.plot_spectrogram.PlotTF method), 192`

`save_figure() (mtpy.modeling.occam1d.Plot1DResponse method), 258`

`save_figure() (mtpy.modeling.occam1d.plot_response.Plot1DResponse method), 245`

`save_figure() (mtpy.modeling.winglink.PlotMisfitPseudoSection method), 320`

`save_figure() (mtpy.modeling.winglink.PlotPseudoSection method), 323`

`save_figures() (mtpy.modeling.winglink.PlotResponse method), 325`

`save_path (mtpy.modeling.modem.Model property), 232`

`save_path (mtpy.modeling.modem.model.Model property), 216`

`save_path (mtpy.modeling.structured_mesh_3d.StructuredGrid3D property), 314`

`save_path (mtpy.modeling.StructuredGrid3D property), 334`

`save_plot() (mtpy.imaging.mplot_tools.base.PlotBase method), 163`

`save_plot() (mtpy.imaging.mplot_tools.PlotBase method), 175`

`ScriptFile (class in mtpy.processing.birrp), 343`

`ScriptFileError, 346`

`select_station_runs()`  
`(mtpy.processing.kernel_dataset.KernelDataset method), 354`

`select_station_runs()`  
`(mtpy.processing.KernelDataset method), 375`

`set_amp_phase() (mtpy.core.Tipper method), 155`

`set_amp_phase() (mtpy.core.transfer_function.Tipper method), 105`

`set_amp_phase() (mtpy.core.transfer_function.tipper.Tipper method), 97`

`set_floor() (mtpy.modeling.errors.ModelErrors method), 304`

`set_mag_direction() (mtpy.core.Tipper method), 155`

`set_mag_direction()`  
`(mtpy.core.transfer_function.Tipper method), 105`

`set_mag_direction()`  
`(mtpy.core.transfer_function.tipper.Tipper method), 97`

`set_path() (mtpy.processing.kernel_dataset.KernelDataset class method), 354`

`set_path() (mtpy.processing.KernelDataset class method), 375`

`set_period_limits()`  
`(mtpy.imaging.mplot_tools.plot_settings.PlotSettings method), 169`

`set_period_limits()`  
`(mtpy.imaging.mplot_tools.PlotSettings method), 177`

`set_phase_limits() (mtpy.imaging.mplot_tools.plot_settings.PlotSettings method), 169`

`set_phase_limits() (mtpy.imaging.mplot_tools.PlotSettings method), 177`

`set_resistivity_limits()`  
`(mtpy.imaging.mplot_tools.plot_settings.PlotSettings method), 169`

`set_resistivity_limits()`  
`(mtpy.imaging.mplot_tools.PlotSettings method), 177`

`set_resistivity_phase()`  
`(mtpy.core.transfer_function.Z method), 110`

`set_resistivity_phase()`  
`(mtpy.core.transfer_function.z.Z method), 101`

`set_resistivity_phase() (mtpy.core.Z method), 160`

`set_rpt() (mtpy.analysis.residual_phase_tensor.ResidualPhaseTensor method), 83`

`set_rpt_error() (mtpy.analysis.residual_phase_tensor.ResidualPhaseTensor method), 83`

`set_run_times() (mtpy.processing.kernel_dataset.KernelDataset method), 354`

method), 355  
set\_run\_times() (mtpy.processing.KernelDataset method), 375  
set\_sample\_rate() (mtpy.processing.run\_summary.RunSummary method), 357  
set\_sample\_rate() (mtpy.processing.RunSummary method), 376  
Sgrid (class in mtpy.modeling.gocad), 305  
ShapefileCreator (class in mtpy.gis.shapefile\_creator), 161  
Simpeg1D (class in mtpy.modeling.simpeg.recipes), 286  
Simpeg1D (class in mtpy.modeling.simpeg.recipes.inversion\_Id), 285  
Simpeg2D (class in mtpy.modeling.simpeg.recipes), 287  
sinc\_filter() (in module mtpy.processing.tf), 365  
size (mtpy.core.mt\_dataframe.MTDataFrame property), 139  
size (mtpy.core.MTDataFrame property), 146  
skew (mtpy.core.PhaseTensor property), 154  
skew (mtpy.core.transfer\_function.PhaseTensor property), 104  
skew (mtpy.core.transfer\_function.pt.PhaseTensor property), 95  
skew\_cmap\_bounds (mtpy.imaging.plot\_phase\_tensor\_maps property), 184  
skew\_cmap\_bounds (mtpy.imaging.PlotPhaseTensorMaps property), 198  
skew\_error (mtpy.core.PhaseTensor property), 154  
skew\_error (mtpy.core.transfer\_function.PhaseTensor property), 104  
skew\_error (mtpy.core.transfer\_function.pt.PhaseTensor property), 95  
skew\_model\_error (mtpy.core.PhaseTensor property), 154  
skew\_model\_error (mtpy.core.transfer\_function.PhaseTensor property), 104  
skew\_model\_error (mtpy.core.transfer\_function.pt.PhaseTensor property), 95  
smethod() (in module mtpy.processing.tf), 365  
sort\_folder\_list() (in module mtpy.utils.filehandling), 388  
specwv() (in module mtpy.processing.tf), 366  
spwvd() (in module mtpy.processing.tf), 367  
starting\_beta (mtpy.modeling.simpeg.recipes.Simpeg2D property), 289  
Startup (class in mtpy.modeling.occam2d), 281  
Startup (class in mtpy.modeling.occam2d.startup), 270  
startup\_fn (mtpy.modeling.ws3dinv.startup.WSStartup property), 295  
startup\_fn (mtpy.modeling.ws3dinv.WSStartup property), 301  
station (mtpy.core.mt\_dataframe.MTDataFrame property), 139  
station (mtpy.core.MTDataFrame property), 146  
station (mtpy.modeling.modem.station.Stations property), 222  
station\_filename (mtpy.modeling.ws3dinv.stations.WSStation property), 297  
station\_filename (mtpy.modeling.ws3dinv.WSStation property), 302  
station\_locations (mtpy.core.mt\_dataframe.MTDataFrame property), 139  
station\_locations (mtpy.core.mt\_stations.MTStations property), 143  
station\_locations (mtpy.core.MTDataFrame property), 146  
station\_locations (mtpy.core.MTStations property), 150  
station\_total\_length (mtpy.modeling.simpeg.make\_2d\_mesh.StructuredMesh property), 291  
Stations (class in mtpy.modeling.modem.station), 221  
stations (mtpy.modeling.occam2d.data.Occam2DData property), 262  
stations (mtpy.modeling.occam2d.Occam2DData property), 278  
stfbss() (in module mtpy.processing.tf), 368  
strike (mtpy.core.transfer\_function.z\_analysis.ZInvariants property), 89  
strike (mtpy.core.transfer\_function.z\_analysis.ZInvariants property), 88  
strike\_error (mtpy.core.transfer\_function.z\_analysis.ZInvariants property), 89  
strike\_error (mtpy.core.transfer\_function.z\_analysis.ZInvariants property), 88  
structure\_3d (mtpy.core.transfer\_function.z\_analysis.ZInvariants property), 89  
structure\_3d (mtpy.core.transfer\_function.z\_analysis.ZInvariants property), 88  
StructuredGrid3D (class in mtpy.modeling), 327  
StructuredGrid3D (class in mtpy.modeling.structured\_mesh\_3d), 307  
StructuredMesh (class in mtpy.modeling.simpeg.make\_2d\_mesh), 291  
summarize\_data\_quality() (mtpy.utils.estimate\_tf\_quality\_factor.EMTFStats method), 384  
survey (mtpy.core.mt\_dataframe.MTDataFrame property), 139  
survey (mtpy.core.MTDataFrame property), 146  
survey\_ids (mtpy.core.mt\_data.MTData property), 134  
survey\_ids (mtpy.MTData property), 417

## T

target\_misfit (mtpy.modeling.simpeg.recipes.Simpeg2D property), 289

`te_data_misfit (mtpy.modeling.simpeg.recipes.Simpeg2D to_dataframe()) (mtpy.core.mt_data.MTData method),  
property), 289  
135`

`te_simulation (mtpy.modeling.simpeg.recipes.Simpeg2D to_dataframe()) (mtpy.core.transfer_function.base.TFBase  
method), 289  
93`

`text_dict (mtpy.imaging.mtplot_tools.plot_settings.PlotSettings to_dataframe() (mtpy.MT method), 399  
property), 169  
to_dataframe() (mtpy.MTData method), 417`

`text_dict (mtpy.imaging.mtplot_tools.PlotSettings to_dict() (mtpy.processing.birrp.BIRRPPParameters  
method), 341  
method), 341`

`TFBase to_geo_df() (mtpy.core.mt_collection.MTCollection  
module, 91  
method), 123`

`TFBase (class in mtpy.core.transfer_function.base), 91  
to_geo_df() (mtpy.core.mt_data.MTData method), 135`

`thicknesses (mtpy.modeling.simpeg.recipes.inversion_1d.  
property), 286  
SingeDF() (mtpy.MTCollection method), 406`

`thicknesses (mtpy.modeling.simpeg.recipes.Simpeg1D  
property), 287  
to_geo_df() (mtpy.MTData method), 417`

`Tipper (class in mtpy.core), 154  
Tipper (class in mtpy.core.transfer_function), 104  
Tipper (class in mtpy.core.transfer_function.tipper), 96  
Tipper (mtpy.core.mt.MT property), 111  
to_geopd() (mtpy.core.mt_stations.MTStations  
tipper (mtpy.core.mt_dataframe.MTDataFrame prop-  
erty), 139  
method), 143`

`tipper (mtpy.core.MTDataFrame property), 146  
to_geopd() (mtpy.core.MTStations method), 150`

`tipper (mtpy.core.transfer_function.Tipper property),  
105  
to_geopd() (mtpy.modeling.modem.station.Stations  
method), 222`

`tipper (mtpy.core.transfer_function.tipper.Tipper prop-  
erty), 97  
to_geosoft_xyz() (mtpy.modeling.structured_mesh_3d.StructuredGrid3D  
Tipper (mtpy.MT property), 394  
method), 315`

`tipper_error (mtpy.core.Tipper property), 155  
to_geosoft_xyz() (mtpy.modeling.StructuredGrid3D  
tipper_error (mtpy.core.transfer_function.Tipper  
property), 105  
method), 335`

`tipper_error (mtpy.core.transfer_function.tipper.Tipper  
property), 97  
to_gocad_sgrid() (mtpy.modeling.structured_mesh_3d.StructuredGrid3D  
tipper_error (mtpy.core.transfer_function.Tipper property), 156  
method), 315`

`tipper_model_error (mtpy.core.Tipper property), 156  
to_gocad_sgrid() (mtpy.modeling.StructuredGrid3D  
tipper_model_error (mtpy.core.transfer_function.Tipper  
property), 106  
method), 335`

`tipper_model_error (mtpy.core.transfer_function.tipper.Tipper  
property), 97  
to_gocad_sgrid() (mtpy.modeling.StructuredGrid3D  
tm_data_misfit (mtpy.modeling.simpeg.recipes.Simpeg2D to_mt_data()  
property), 290  
method), 123`

`tm_simulation (mtpy.modeling.simpeg.recipes.Simpeg2D to_mt_data() (mtpy.MTCollection method), 406  
property), 290  
to_modem() (mtpy.core.mt_data.MTData method), 135`

`to_modem() (mtpy.modeling.structured_mesh_3d.StructuredGrid3D  
tipper (mtpy.core.mt_dataframe.MTDataFrame method), 418  
method), 418`

`tipper (mtpy.core.transfer_function.tipper.Tipper  
method), 135  
to_modem() (mtpy.MTData method), 418`

`tm_data_misfit (mtpy.modeling.simpeg.recipes.Simpeg2D to_mt_data() (mtpy.core.mt_collection.MTCollection  
property), 290  
method), 123`

`tm_simulation (mtpy.modeling.simpeg.recipes.Simpeg2D to_mt_data() (mtpy.MTCollection method), 406  
property), 290  
to_modem() (mtpy.core.mt_data.MTData method), 135`

`to_conductance_raster()  
(mtpy.modeling.structured_mesh_3d.StructuredGrid3D  
method), 314  
to_modem() (mtpy.core.mt_dataframe.MTDataFrame method), 418`

`to_conductance_raster()  
(mtpy.modeling.StructuredGrid3D  
method), 334  
to_modem() (mtpy.MTData method), 418`

`to_csv() (mtpy.core.mt_stations.MTStations method),  
143  
to_modem() (mtpy.modeling.structured_mesh_3d.StructuredGrid3D  
method), 316`

`to_csv() (mtpy.core.MTStations method), 150  
to_modem() (mtpy.modeling.structured_mesh_3d.StructuredGrid3D  
method), 316`

`to_csv() (mtpy.modeling.modem.station.Stations  
method), 222  
to_modem() (mtpy.MTData method), 418`

`to_dataframe() (mtpy.core.mt.MT method), 116  
to_modem() (mtpy.core.mt_dataframe.MTDataFrame  
method), 418`

`to_occam1d() (mtpy.core.mt.MT method), 117  
to_occam1d() (mtpy.MT method), 399  
to_occam2d() (mtpy.core.mt_data.MTData method),  
136  
to_occam2d() (mtpy.MTData method), 418  
to_occam2d_data() (mtpy.core.mt_data.MTData  
method), 418`

*method*), 136  
to\_occam2d\_data() (*mtpy.MTData method*), 418  
to\_raster() (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D method*), 316  
to\_raster() (*mtpy.modeling.StructuredGrid3D method*), 336  
to\_shp() (*mtpy.core.mt\_collection.MTCollection method*), 123  
to\_shp() (*mtpy.core.mt\_stations.MTStations method*), 143  
to\_shp() (*mtpy.core.MTStations method*), 151  
to\_shp() (*mtpy.modeling.modem.station.Stations method*), 222  
to\_shp() (*mtpy.MTCollection method*), 406  
to\_shp\_pt\_tipper() (*mtpy.core.mt\_data.MTData method*), 136  
to\_shp\_pt\_tipper() (*mtpy.MTData method*), 418  
to\_simpeg\_1d() (*mtpy.core.mt.MT method*), 117  
to\_simpeg\_1d() (*mtpy.MT method*), 399  
to\_simpeg\_2d() (*mtpy.core.mt\_data.MTData method*), 137  
to\_simpeg\_2d() (*mtpy.MTData method*), 419  
to\_simpeg\_3d() (*mtpy.core.mt\_data.MTData method*), 137  
to\_simpeg\_3d() (*mtpy.MTData method*), 419  
to\_t\_object() (*mtpy.core.mt\_dataframe.MTDataFrame method*), 139  
to\_t\_object() (*mtpy.core.MTDataFrame method*), 146  
to\_ubc() (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D method*), 317  
to\_ubc() (*mtpy.modeling.StructuredGrid3D method*), 337  
to\_vtk() (*mtpy.core.mt\_stations.MTStations method*), 144  
to\_vtk() (*mtpy.core.MTStations method*), 151  
to\_vtk() (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D method*), 317  
to\_vtk() (*mtpy.modeling.StructuredGrid3D method*), 337  
to\_winglink\_out() (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D method*), 318  
to\_winglink\_out() (*mtpy.modeling.StructuredGrid3D method*), 338  
to\_ws3dinv\_intial() (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D method*), 318  
to\_ws3dinv\_intial() (*mtpy.modeling.StructuredGrid3D method*), 338  
to\_xarray() (*mtpy.core.transfer\_function.base.TFBase method*), 93  
to\_xarray() (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D method*), 318  
to\_xarray() (*mtpy.modeling.StructuredGrid3D*     *method*), 338  
    *method*), 338  
to\_xyres() (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D method*), 318  
to\_xyres() (*mtpy.modeling.StructuredGrid3D method*), 338  
to\_xyzres() (*mtpy.modeling.structured\_mesh\_3d.StructuredGrid3D method*), 318  
to\_xyzres() (*mtpy.modeling.StructuredGrid3D method*), 338  
to\_z\_object() (*mtpy.core.mt\_dataframe.MTDataFrame method*), 139  
to\_z\_object() (*mtpy.core.MTDataFrame method*), 146  
trace (*mtpy.core.PhaseTensor property*), 154  
trace (*mtpy.core.transfer\_function.PhaseTensor property*), 104  
trace (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 95  
trace\_error (*mtpy.core.PhaseTensor property*), 154  
trace\_error (*mtpy.core.transfer\_function.PhaseTensor property*), 104  
trace\_error (*mtpy.core.transfer\_function.pt.PhaseTensor property*), 95  
trace\_model\_error (*mtpy.core.PhaseTensor property*), 154  
trace\_model\_error (*mtpy.core.transfer\_function.PhaseTensor property*), 104  
triangulate\_interpolation() (*in module* *mtpy.imaging.mplot\_tools*), 180  
triangulate\_interpolation() (*in module* *mtpy.imaging.mplot\_tools.map\_interpolation\_tools*), 168  
tukey() (*in module* *mtpy.processing.filter*), 349

## U

*units* (*mtpy.core.transfer\_function.Z property*), 110  
    *units* (*mtpy.core.transfer\_function.z.Z property*), 102  
    *units* (*mtpy.core.Z property*), 160  
update\_plot() (*mtpy.imaging.mplot\_tools.base.PlotBase method*), 164  
update\_plot() (*mtpy.imaging.mplot\_tools.PlotBase method*), 176  
update\_plot() (*mtpy.imaging.plot\_spectrogram.PlotTF method*), 193  
update\_plot() (*mtpy.modeling.occam1d.Plot1DResponse method*), 259  
update\_plot() (*mtpy.modeling.occam1d.plot\_response.Plot1DResponse method*), 246  
update\_plot() (*mtpy.modeling.winglink.PlotMisfitPseudoSection method*), 321  
update\_plot() (*mtpy.modeling.winglink.PlotPseudoSection method*), 324

update\_survey\_metadata()  
     (*mtpy.processing.kernel\_dataset.KernelDataset method*), 355

update\_survey\_metadata()  
     (*mtpy.processing.KernelDataset method*), 376

use\_measurement\_error()  
     (*mtpy.modeling.errors.ModelErrors method*), 304

utm\_crs (*mtpy.core.mt\_location.MTLocation property*), 140

utm\_crs (*mtpy.core.mt\_stations.MTStations property*), 144

utm\_crs (*mtpy.core.MTLocation property*), 148

utm\_crs (*mtpy.core.MTStations property*), 151

utm\_epsg (*mtpy.core.mt\_dataframe.MTDataFrame property*), 139

utm\_epsg (*mtpy.core.mt\_location.MTLocation property*), 140

utm\_epsg (*mtpy.core.mt\_stations.MTStations property*), 144

utm\_epsg (*mtpy.core.MTDataFrame property*), 146

utm\_epsg (*mtpy.core.MTLocation property*), 148

utm\_epsg (*mtpy.core.MTStations property*), 151

utm\_name (*mtpy.core.mt\_location.MTLocation property*), 140

utm\_name (*mtpy.core.mt\_stations.MTStations property*), 144

utm\_name (*mtpy.core.MTLocation property*), 148

utm\_name (*mtpy.core.MTStations property*), 152

utm\_zone (*mtpy.core.mt\_location.MTLocation property*), 140

utm\_zone (*mtpy.core.mt\_stations.MTStations property*), 144

utm\_zone (*mtpy.core.MTLocation property*), 148

utm\_zone (*mtpy.core.MTStations property*), 152

utm\_zone (*mtpy.modeling.modem.station.Stations property*), 222

**V**

validate\_array\_shape()  
     (*mtpy.modeling.errors.ModelErrors method*), 304

validate\_input\_values()     (*in module mtpy.utils.gis\_tools*), 390

validate\_percent() (*mtpy.modeling.errors.ModelErrors method*), 305

validate\_save\_file()     (*in module mtpy.utils.filehandling*), 388

validate\_ts\_file()     (*in module mtpy.utils.filehandling*), 389

**W**

WLInputError, 326

working\_directory (*mtpy.core.mt\_collection.MTCollection property*), 124

working\_directory (*mtpy.MTCollection property*), 406

write\_config\_file()  
     (*mtpy.modeling.modem.config.ModEMConfig method*), 206

write\_config\_file()  
     (*mtpy.modeling.modem.ModEMConfig method*), 227

write\_config\_file()  
     (*mtpy.processing.birrp.BIRRPParameters method*), 341

write\_config\_from\_survey\_txt\_file() (*in module mtpy.utils.configfile*), 382

write\_control\_file()  
     (*mtpy.modeling.modem.control\_fwd.ControlFwd method*), 207

write\_control\_file()  
     (*mtpy.modeling.modem.control\_inv.ControlInv method*), 208

write\_control\_file()  
     (*mtpy.modeling.modem.ControlFwd method*), 223

write\_control\_file()  
     (*mtpy.modeling.modem.ControlInv method*), 224

write\_cov\_vtk\_file()  
     (*mtpy.modeling.modem.convariance.Covariance method*), 208

write\_cov\_vtk\_file()  
     (*mtpy.modeling.modem.Covariance method*), 225

write\_covariance\_file()  
     (*mtpy.modeling.modem.convariance.Covariance method*), 208

write\_covariance\_file()  
     (*mtpy.modeling.modem.Covariance method*), 225

write\_data\_file()     (*mtpy.modeling.modem.Data method*), 226

write\_data\_file()     (*mtpy.modeling.modem.data.Data method*), 210

write\_data\_file()     (*mtpy.modeling.occam1d.data.Occam1DData method*), 239

write\_data\_file()     (*mtpy.modeling.occam1d.Occam1DData method*), 251

write\_data\_file()     (*mtpy.modeling.occam2d.data.Occam2DData method*), 262

write\_data\_file()     (*mtpy.modeling.occam2d.Occam2DData method*), 278

write\_data\_file()     (*mtpy.modeling.ws3dinv.data.WSData method*), 294

write\_data\_file()     (*mtpy.modeling.ws3dinv.WSData method*), 294

method), 300  
write\_dict\_to\_configfile() (in module mtpy.utils.configfile), 383  
write\_edi\_file() (mtpy.processing.birrp.J2Edi method), 342  
write\_geosoft\_xyz() (mtpy.modeling.modem.Model method), 232  
write\_geosoft\_xyz() (mtpy.modeling.modem.model.Model method), 216  
write\_gocad\_sgrid\_file() (mtpy.modeling.modem.Model method), 233  
write\_gocad\_sgrid\_file() (mtpy.modeling.modem.model.Model method), 217  
write\_iter\_file() (mtpy.modeling.occam2d.model.Occam2DModel method), 268  
write\_iter\_file() (mtpy.modeling.occam2d.Occam2DModel method), 279  
write\_mesh\_file() (mtpy.modeling.occam2d.Mesh method), 276  
write\_mesh\_file() (mtpy.modeling.occam2d.mesh.Mesh method), 266  
write\_model\_file() (mtpy.modeling.modem.Model method), 233  
write\_model\_file() (mtpy.modeling.modem.model.Model method), 217  
write\_model\_file() (mtpy.modeling.occam1d.model.Occam1DModel method), 297  
write\_model\_file() (mtpy.modeling.occam1d.Occam1DModel method), 242  
write\_model\_file() (mtpy.modeling.modem.model.Model method), 254  
write\_out\_file() (mtpy.modeling.modem.Model method), 234  
write\_out\_file() (mtpy.modeling.modem.model.Model method), 218  
write\_regularization\_file() (mtpy.modeling.occam2d.Regularization method), 281  
write\_regularization\_file() (mtpy.modeling.occam2d.regularization.Regularization method), 270  
write\_script\_file() (mtpy.processing.birrp.ScriptFile method), 346  
write\_sgrid\_file() (mtpy.modeling.gocad.Sgrid method), 305  
write\_startup\_file() (mtpy.modeling.occam1d.Occam1DStartup method), 255  
write\_startup\_file() (mtpy.modeling.occam1d.startup.Occam1DStartup method), 248  
write\_startup\_file()

(mtpy.modeling.occam2d.Startup method), 282  
write\_startup\_file() (mtpy.modeling.occam2d.startup.Startup method), 271  
write\_startup\_file() (mtpy.modeling.ws3dinv.startup.WSStartup method), 296  
write\_startup\_file() (mtpy.modeling.ws3dinv.WSStartup method), 301  
write\_station\_file() (mtpy.modeling.ws3dinv.stations.WSStation method), 297  
write\_station\_file() (mtpy.modeling.ws3dinv.WSStation method), 217  
write\_ts\_file\_from\_tuple() (in module mtpy.utils.filehandling), 389  
write\_ubc\_files() (mtpy.modeling.modem.Model method), 234  
write\_ubc\_files() (mtpy.modeling.modem.model.Model method), 218  
write\_vtk\_file() (mtpy.modeling.modem.Model method), 234  
write\_vtk\_file() (mtpy.modeling.modem.model.Model method), 218  
write\_vtk\_file() (mtpy.modeling.ws3dinv.stations.WSStation method), 297  
write\_vtk\_file() (mtpy.modeling.ws3dinv.WSStation method), 303  
write\_xyres() (mtpy.modeling.modem.Model method), 235  
write\_xyres() (mtpy.modeling.modem.model.Model method), 219  
write\_xyzres() (mtpy.modeling.modem.Model method), 235  
write\_xyzres() (mtpy.modeling.modem.model.Model method), 219  
WSData (class in mtpy.modeling.ws3dinv), 297  
WSData (class in mtpy.modeling.ws3dinv.data), 292  
WSStartup (class in mtpy.modeling.ws3dinv), 301  
WSStartup (class in mtpy.modeling.ws3dinv.startup), 295  
WSStation (class in mtpy.modeling.ws3dinv), 301  
WSStation (class in mtpy.modeling.ws3dinv.stations), 296  
wvd() (in module mtpy.processing.if), 369  
wvd\_analytic\_signal() (in module mtpy.processing.tf), 369

X  
x\_key (mtpy.gis.shapefile\_creator.ShapefileCreator property), 162

<code>x_pad (mtpy.modeling.simpeg.make_2d_mesh.QuadTreeMesh.x_pad property), 290</code>	<code>pad_down (mtpy.modeling.simpeg.make_2d_mesh.QuadTreeMesh.property), 291</code>
<code>x_padding_cells (mtpy.modeling.simpeg.make_2d_mesh.StructuredMesh.x_padding_cells property), 291</code>	<code>structuredMesh (mtpy.modeling.simpeg.make_2d_mesh.QuadTreeMesh.property), 291</code>
<code>x_total (mtpy.modeling.simpeg.make_2d_mesh.QuadTreeMesh.x_total property), 290</code>	<code>total (mtpy.modeling.simpeg.make_2d_mesh.QuadTreeMesh.property), 291</code>
<code>xy_error_bar_properties (mtpy.imaging.mplot_tools.plot_settings.PlotSettings.xy_error_bar_properties property), 169</code>	<code>zero_pad() (in module mtpy.processing.filter), 349</code>
<code>xy_error_bar_properties (mtpy.imaging.mplot_tools.PlotSettings.xy_error_bar_properties property), 177</code>	<code>ZInvariants (class in mtpy.core.transfer_function.z_analysis), 88</code>
	<code>ZInvariants (class in mtpy.core.transfer_function.z_analysis.zinvariants), 87</code>

**Y**

<code>y_key (mtpy.gis.shapefile_creator.ShapefileCreator.y_key property), 162</code>
<code>yx_error_bar_properties (mtpy.imaging.mplot_tools.plot_settings.PlotSettings.yx_error_bar_properties property), 169</code>
<code>yx_error_bar_properties (mtpy.imaging.mplot_tools.PlotSettings.yx_error_bar_properties property), 177</code>

**Z**

<code>Z (class in mtpy.core), 156</code>
<code>Z (class in mtpy.core.transfer_function), 106</code>
<code>Z (class in mtpy.core.transfer_function.z), 97</code>
<code>Z (mtpy.core.mt.MT property), 111</code>
<code>z (mtpy.core.transfer_function.Z property), 110</code>
<code>z (mtpy.core.transfer_function.z.Z property), 102</code>
<code>z (mtpy.core.Z property), 160</code>
<code>Z (mtpy.MT property), 394</code>
<code>z1_layer_thickness (mtpy.modeling.simpeg.make_2d_mesh.StructuredMesh.z1_layer_thickness property), 291</code>
<code>z_bottom (mtpy.modeling.simpeg.make_2d_mesh.StructuredMesh.z_bottom property), 291</code>
<code>z_core (mtpy.modeling.simpeg.make_2d_mesh.QuadTreeMesh.z_core property), 290</code>
<code>z_error (mtpy.core.transfer_function.Z property), 110</code>
<code>z_error (mtpy.core.transfer_function.z.Z property), 102</code>
<code>z_error (mtpy.core.Z property), 160</code>
<code>z_error2r_phi_error() (in module mtpy.utils.calculator), 380</code>
<code>z_max (mtpy.modeling.simpeg.make_2d_mesh.QuadTreeMesh.z_max property), 291</code>
<code>z_mesh_down (mtpy.modeling.simpeg.make_2d_mesh.StructuredMesh.z_mesh_down property), 291</code>
<code>z_mesh_up (mtpy.modeling.simpeg.make_2d_mesh.StructuredMesh.z_mesh_up property), 291</code>
<code>z_model_error (mtpy.core.transfer_function.Z property), 110</code>
<code>z_model_error (mtpy.core.transfer_function.z.Z property), 102</code>
<code>z_model_error (mtpy.core.Z property), 160</code>