Aurora: An open-source python implementation of the EMTF package for magnetotelluric data processing using MTH5 and mt_metadata

Karl N. Kappler[5], **Jared R. Peacock**[1], **Gary D. Egbert**[2], **Andrew Frassetto**[3], **Lindsey Heagy**[4], **Anna Kelbert**[1], **Laura Keyson**[3], **Douglas Oldenburg**[4], **Timothy Ronan**[3], and **Justin Sweet**[3]

1 United States Geological Survey, USA **2** Oregon State University, USA **3** Earthscope, USA **4** University of British Columbia, USA **5** Independent Researcher, USA

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

The Aurora software package robustly estimates single station and remote reference electromagnetic transfer functions (TFs) from magnetotelluric (MT) time series. Aurora is part of an open-source processsing workflow that leverages the self-describing data container MTH5, which in turn leverages the general mt_metadata framework to manage metadata. These pre-existing tools greatly simplify the processing interface, reducing requirements for specialized domain knowledge in time series analysis, or data structures manangement and generating transfer functions with a few lines of code. The processing depends on two inputs – a table specifying the data to use for TF estimation, and a JSON file specfiying the processing parameters, both of which are generated automatically, and can be modified if desired. Output TFs are returned as mt_metadata objects, and can be exported to a variety of common formats for plotting, modelling and inversion.

Introduction

Magnetotellurics (MT) is a geophysical technique for probing the electrical conductivity structure of the subsurface using co-located electric and magnetic field measurements. After data collection, standard practice is to estimate the time invariant (frequency domain) transfer function (TF) between electric and magnetic channels before proceeding to interpretation and modelling. If measurements are orthogonal the TF is equivlent to the electrical impedance tensor (Z) (Vozoff, 1991).

$$\begin{bmatrix} E_x \\ E_y \end{bmatrix} = \begin{bmatrix} Z_{xx} & Z_{xy} \\ Z_{yx} & Z_{yy} \end{bmatrix} \begin{bmatrix} H_x \\ H_y \end{bmatrix}$$

where (E_x, E_y), (H_x, H_y) denote orthogonal electric and magnetic fields respectively. TF estimation involves management of metadata (locations, orientations, timestamps,) versatile data containers (for linear algebra, slicing, plotting, etc.) and uses a broad collection of signal processing and statistical techniques (Egbert (1997) and references therein). MTH5 supplies time series as xarray objects for efficient, lazy access to data and easy application of linear algebra and statistics libraries available in the python.

Statement of Need

Uncompiled FORTRAN processing codes have been available for years (e.g. EMTF Egbert et al. (2017), or BIRRP Chave (1989)) but do not offer the readability of a high-level languge and modifications are seldom attempted (Egbert et al., 2017). Recently several python versions of MT processing codes have been released by the open source community, including Shah et al. (2019), Smaï & Wawrzyniak (2020), Ajithabh & Patro (2023), and Friedrichs (2022). Aurora adds to this canon of options but differs by leveraging the MTH5 and mt_metadata packages elimiating a need for internal development of time series or metadata containers. By providing an example workflow employing mt_metadata and MTH5 we hope other developers may benefit from following this model, allowing researchers interested in signal-and-noise separation in MT to spend more time exploring and testing algorithms to improve TF estimates, and less time (re)-developing formats and management tools for data and metadata. As a python representation of Egbert's EMTF Remote Reference processing software, Aurora also provides a sort of continuity in the code space as the languages evolve.

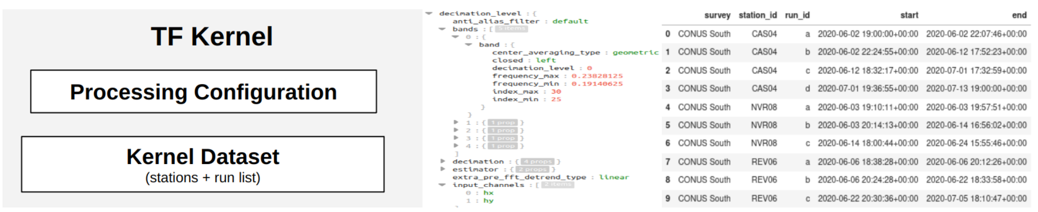
This manuscript describes the high-level concepts of the software – for information about MT data processing Ajithabh & Patro (2023) provides a concise summary, and more in-depth details can be found in Vozoff (1991), Egbert (2002) and references therein.

Key Features

- Tabular Data indexing and management (pandas data frames),
- Dicitionary-like Processing parameters configuration
- Both allow for programatic or manual editting.

A TF instance depends on two key prior decisions: a) The data input to the TF computation algorithm, b) The algorithm itself including the specific values of the processing parameters. Aurora formalizes these concepts as classes (KernelDataset and Processing, respectively), and a third class TransferFunctionKernel (TFK Figure 1), a compositon of the Processing, and KernelDataset provides a place for logic validating consistency between selected data and processing parameters. TFK specifies all the information needed to make the calculation of a TF reproducible (supporting the R in FAIRly archived TFs).

Generation of robust TFs can be done in only a few lines starting from an MTH5 archive (Figure 3). Simplicity of workflow is due to the MTH5 container already storing comprehensive metadata, including a channel summary table describing all time series stored in the archive including start/end times and sample rates. Users can easily view a tabular summary of available data and select station pairs to process. Once a station – and optionally a remote reference station – are defined, the simultaneous time intervals of data coverage at both stations are idenitified automatically, providing the Kernel Dataset. Reasonable starting processing parameters can be automatically generated for a given Kernel Dataset, and editted programatically or via a JSON file. Once the TFK is defined, the processing automatically follows the flow of Figure 2.



```
decimation_level : {
   anti_alias_filter : default
   bands : [ 5 items
      0 : {
         band : {
            center_averaging_type : geometric
            closed : left
            decimation_level : 0
            frequency_max : 0.23828125
            frequency_min : 0.19140625
            index_max : 30
            index_min : 25
         }
      }
      1 : { 1 prop }
      2 : { 1 prop }
      3 : { 1 prop }
      4 : { 1 prop }
   ]
   decimation : { 4 props }
   estimator : { 2 props }
   extra_pre_fft_detrend_type : linear
   input_channels : [ 2 items
      0 : hx
      1 : hy
```

	survey	station_id	run_id	start	end
0	CONUS South	CAS04	a	2020-06-02 19:00:00+00:00	2020-06-02 22:07:46+00:00
1	CONUS South	CAS04	b	2020-06-02 22:24:55+00:00	2020-06-12 17:52:23+00:00
2	CONUS South	CAS04	c	2020-06-12 18:32:17+00:00	2020-07-01 17:32:59+00:00
3	CONUS South	CAS04	d	2020-07-01 19:36:55+00:00	2020-07-13 19:00:00+00:00
4	CONUS South	NVR08	a	2020-06-03 19:10:11+00:00	2020-06-03 19:57:51+00:00
5	CONUS South	NVR08	b	2020-06-03 20:14:13+00:00	2020-06-14 16:56:02+00:00
6	CONUS South	NVR08	c	2020-06-14 18:00:44+00:00	2020-06-24 15:55:46+00:00
7	CONUS South	REV06	a	2020-06-06 18:38:28+00:00	2020-06-06 20:12:26+00:00
8	CONUS South	REV06	b	2020-06-06 20:24:28+00:00	2020-06-22 18:33:58+00:00
9	CONUS South	REV06	c	2020-06-22 20:30:36+00:00	2020-07-05 18:10:47+00:00

Figure 1: TF Kernel concept diagram: A box representing the TF Kernel with two inlay boxes representing the processing config (JSON) and dataset (pandas DataFrame).

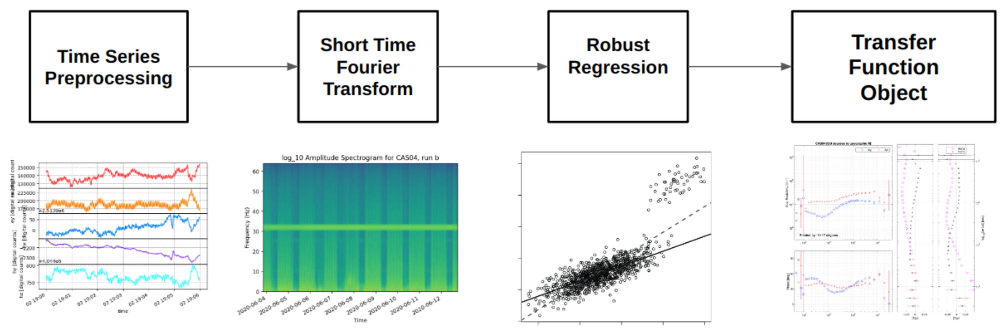

Time Series Preprocessing
Short Time Fourier Transform
Robust Regression
Transfer Function Object
log_10 Amplitude Spectrogram for CAS04, run b

Figure 2: The main interfaces of Aurora TF processing. Example time series from mth5 archive in linked notebook (using MTH5 built-in time series plotting), spectrogram from FC data structure, cartoon from Hand (2018) and TF from SPUD.

Examples

Here an example of the aurora data processing flow is given, using data from Earthscope. This section refers to Jupyter notebooks intended as companions to this paper. A relatively general notebook about accessing Earthscope data with MTH5 can be found in the link from row 1 of Table 1.

Table 1: Referenced jupyter notebooks with links.

ID	Link
1	earthscope_magnetic_data_tutorial
2	make_mth5_driver_v0.2.0
3	process_cas04_mulitple_station

The MTH5 dataset can be built by executing the example notebook in row 2 of Table 1.The data processing can be executed by following the tutorial in row 3 of Table 1 – a condensed version of which is shown in Figure 3. Resultant apparent resistivities are plotted in Figure 4 along with the results hosted at Earthscope from EMTF.

```python
from aurora.config.config_creator import ConfigCreator
from aurora.pipelines.process_mth5 import process_mth5
from aurora.pipelines.run_summary import RunSummary
from aurora.transfer_function.kernel_dataset import KernelDataset
```

```python
run_summary = RunSummary()
run_summary.from_mth5s(["8P_CAS04_NVR08.h5",])
kernel_dataset = KernelDataset()
kernel_dataset.from_run_summary(run_summary, "CAS04", "NVR08")
cc = ConfigCreator()
config = cc.create_from_kernel_dataset(kernel_dataset)
tf = process_mth5(config, kernel_dataset)
tf.write(fn="CAS04_rrNVR08.edi", file_type="edi")
```

Figure 3: Code snippet with steps to generate a TF from an MTH5 (generated by row 1 of Table 1). With MTH5 in present working directory, a table of available contiguous blocks of multichannel time series is generated ("RunSummary"), then station(s) to process are selected (by inspection of the RunSummary dataframe) to generate a KernelDataset. The KernelDataset identifies simultaneous data at the local and reference site, and generates processing parameters, which can be editted before passing them to process_mth5, and finally exporting TF to a standard output format, in this case `edi`.

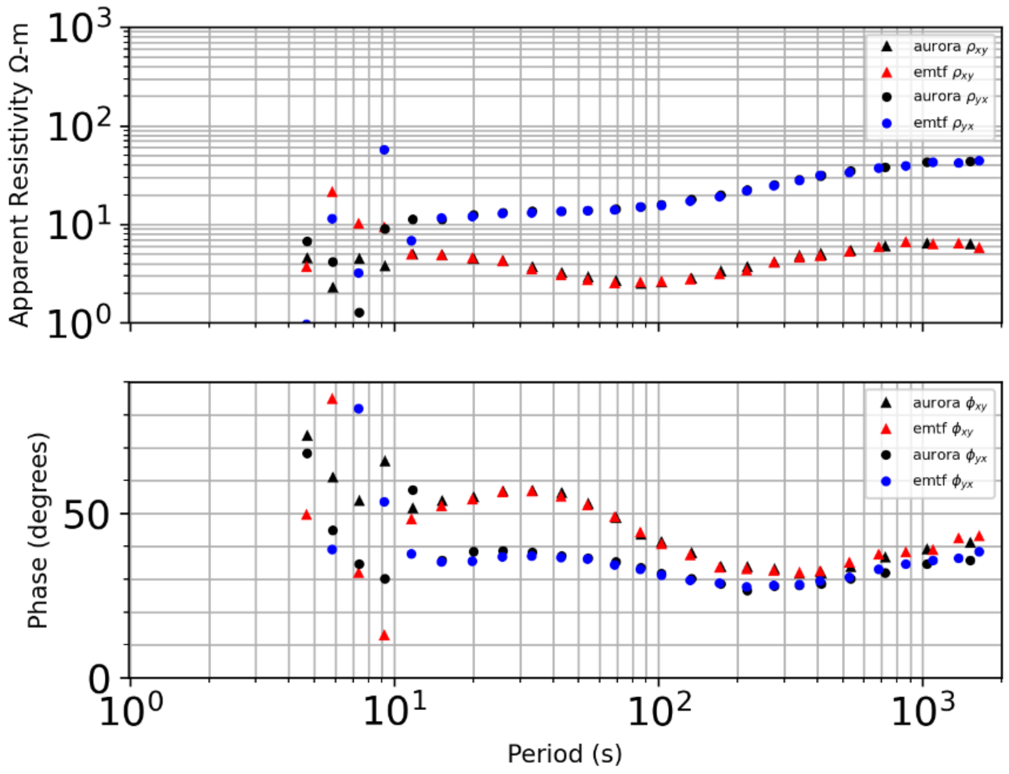


Figure 4: Comparison of apparent resistivities from Aurora and EMTF for station CAS04. Both curves exhibit scatter in the low SNR MT "dead band" beween 1-10s, but most of estimates are very similar.

Testing

The Aurora package uses continuous integration (Duvall et al., 2007) and implements both unit tests as well as integrated tests with 77% code coverage as measured by CodeCov (core dependencies mt_metadata and MTH5 at 84% and 60% respectively). Improvement of test coverage is ongoing. For integreated tests Aurora uses a small synthetic MT dataset originally from EMTF. A few processing configurations with manually validated results are stored in the repository. Deviation from these results causes tests to fail, alerting developers if a code change resulted in an unexpected baseline processing result. In the summer of 2023, widescale testing on Earthscope data archives was performed and showed that the TF results of auora are similar to those form the EMTF fortran codes, in this case for hundreds of real stations rather than a few synthetic ones. Before PyPI, and conda forge releases, example Jupyter notebooks are also run via github actions.

Future Work

Aurora uses github issues to track tasks and planned improvments. In the near future we want to add noise suppression techniques, for example coherence and polarization sorting and Mahalanobis distance (e.g. Ajithabh & Patro (2023), Platz & Weckmann (2019)). We would also like to develop, or plug into a graphical data selection/rejection interface with time series plotting. Besides these improvments to TF quality, we also would like to embed the TFKernel information into both the MTH5 and the output EMTF_XML (Kelbert (2020)). Unit and integrated tests should be expanded, including a test dataset from audio MT band (most test data is sampled at 1Hz). Aurora will continue to codevelop with mt_metadata, MTH5 and MTPy to maintain the ability to provide outputs for inversion and modelling. Ideally the community can participate in a comparative analysis of the opensource codes available to build a recipe book for handling noise from various open-archived datasets.

Conclusion

Aurora provides an open-source Python implementation of the EMTF package for magnetotelluric data processing. Aurora is a prototype worked example of how to plug processing into an existing opensource data and metadata ecosystem (MTH5, mt_metadata, & MTpy). We hope Aurora can be used as an example interface to these packages for the open source MT community, and that these tools will contribute to workflows which can focus more on geoscience analysis, and less on the nuances of data management.

Appendix

Installation Instructions

The package is installable via the Python Package Index (PyPI) as well as via conda-forge.

The installation in pip: > pip install aurora

And via conda forge: > conda install aurora

Documentation

Documentation is hosted by SimPEG Cockett et al. (2015) and can be found at this link

Licence

Aurora is distributed under the MIT open-source licence.

Acknowledgments

The authors would like to thank IRIS (now Earthscope) for supporting the development of Aurora. Joe Capriotti at SimPEG helped with online documentation and the initial release.

TODO:

- ☐ **Update links to ipynb to release branches after mth5/aurora releases.**
- ☐ remove draft watermark
- ☐ Link these issues to discussion in future work? https://github.com/kujaku11/mth5/issues/179, https://github.com/kujaku11/mt_metadata/issues/195

Ajithabh, K., & Patro, P. K. (2023). SigMT: An open-source python package for magnetotelluric data processing. *Computers & Geosciences*, *171*, 105270.

Chave, A. D. (1989). BIRRP: Bounded influence, remote reference processing. *Journal of Geophysical Research*, *94*(B10), 14–215.

Cockett, R., Kang, S., Heagy, L. J., Pidlisecky, A., & Oldenburg, D. W. (2015). SimPEG: An open source framework for simulation and gradient based parameter estimation in geophysical applications. *Computers & Geosciences*.

Duvall, P. M., Matyas, S., & Glover, A. (2007). *Continuous integration: Improving software quality and reducing risk*. Pearson Education.

Egbert, G. D. (1997). Robust multiple-station magnetotelluric data processing. *Geophysical Journal International*, *130*(2), 475–496.

Egbert, G. D. (2002). Processing and interpretation of electromagnetic induction array data. *Surveys in Geophysics*, *23*(2-3), 207–249.

Egbert, G. D., Kelbert, A., & Meqbel, N. M. (2017). Mod3DMT and EMTF: Free software for MT data processing and inversion. *AGU Fall Meeting Abstracts*, *2017*, NS44A–04.

Friedrichs, B. (2022). MTHotel. In *GitHub repository*. GitHub. https://github.com/bfrmtx/MTHotel

Hand, D. J. (2018). Statistical challenges of administrative and transaction data. *Journal of the Royal Statistical Society Series A: Statistics in Society*, *181*(3), 555–605.

Kelbert, A. (2020). EMTF XML: New data interchange format and conversion tools for electromagnetic transfer functions. *Geophysics*, *85*(1), F1–F17.

Platz, A., & Weckmann, U. (2019). An automated new pre-selection tool for noisy magnetotelluric data using the mahalanobis distance and magnetic field constraints. *Geophysical Journal International*, *218*(3), 1853–1872.

Shah, N., Samrock, F., & Saar, M. O. (2019). Resistics: A versatile native python 3 package for processing of magnetotelluric data. *28. Schmucker-Weidelt-Kolloquium für Elektromagnetische Tiefenforschung*.

Smaï, F., & Wawrzyniak, P. (2020). Razorback, an open source python library for robust processing of magnetotelluric data. *Frontiers in Earth Science*, *8*, 296.

Vozoff, K. (1991). *The Magnetotelluric Method*. https://pubs.geoscienceworld.org/seg/books/book/2087/chapter-abstract/114406941/THE-MAGNETOTELLURIC-METHOD?redirectedFrom=fulltext