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

Karl N. Kappler [5,6], **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 U.S. Geological Survey, USA **2** Oregon State University, USA **3** EarthScope, USA **4** University of British Columbia, USA **5** Space Science Institute, USA **6** DIAS Geophysical, Canada

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 processing workflow that leverages the self-describing data container MTH5, which in turn leverages the general mt_metadata framework to manage metadata. These pre-existing packages simplify the processing workflow by providing managed data structures, transfer functions to be generated with only a few lines of code. The processing depends on two inputs – a table defining the data to use for TF estimation and a JSON file specifying 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, modeling, and inversion.

Key Features

- Tabular data indexing and management (Pandas dataframes),
- Dictionary-like processing parameters configuration
- Programmatic or manual editing of inputs
- Largely automated workflow

Introduction

Magnetotellurics (MT) is a geophysical technique for probing subsurface electrical conductivity using collocated electric and magnetic field measurements. Field data is collected in the time domain, however the Earth can be approximated as a linear system in the frequency domain. Therefore, common practice is to estimate the time invariant (frequency domain) transfer function (TF) between electric and magnetic channels to get information of the Earth's resistivity structure (Egbert, 2002). If measurements are orthogonal, the TF is equivalent 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 requires the E and H time series *and* metadata (locations, orientations, timestamps) along with a collection of signal processing and statistical techniques (Egbert (1997) and references therein). The MTH5 data container archives metadata *with* the data (Peacock et

al. ([2022](#))) and supplies time series as xarray (Hoyer & Hamman ([2017](#))) objects for efficient, lazy access to data and easy application of scientific computing libraries available in Python.

Statement of Need

FORTRAN processing codes have long been available (e.g. EMTF Egbert et al. ([2017](#)), or BIRRP Chave ([1989](#))) but lack the readability of high-level languages and modifications to these programs are seldom attempted ([Egbert et al., 2017](#)), and have the additional barrier of compiling. 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 eliminating a need for development of time series or metadata containers (Peacock et al. ([2022](#))). As a Python representation of Egbert's EMTF Remote Reference processing software, Aurora provides a continuity in the MT code space as the languages evolve. Aurora is two degrees separated from the FORTRAN EMTF, as we used a Matlab implementation of EMTF from Prof. from Gary Egbert (Oregon State University, written communication, 2022-05-01) as an initial framework. By providing an example workflow employing 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 developing and redeveloping formats and management tools for data and metadata. Aurora is distributed under the [MIT](#) open-source license.

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.

Problem Approach

A TF instance depends on two key prior decisions: a) The data input to the TF computation algorithm, and 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` (Figure [1](#)), a composition of the `Processing`, and `KernelDataset`. `TransferFunctionKernel` provides a place for validating consistency between selected data and processing parameters and specifies all information needed to make the calculation of a TF reproducible.

Generation of robust TFs can be done in only a few lines starting from an MTH5 archive. Simplicity of workflow is due to the MTH5 data 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 identified automatically, providing the `KernelDataset`. Reasonable starting processing parameters are automatically generated for a given `KernelDataset`, and can be modified with code or via manual changes to a JSON file. Once the `TransferFunctionKernel` is defined, the processing automatically follows the flow described by Figure [2](#). Input time series are from a MTH5, these can initially be drawn from Phoenix, LEMI, FDSN, Metronix, Zonge, systems etc. and the resultant transfer functions can be exported to the most common TF formats such as .edi, .zmm, ,j, .avg, .xml etc. The images of Figure [2](#) are conceptual – in reality the time series can have data from more than one station, and the spectrograms are also multivariate (not single channel as shown). The regression is also multivariate, and applied on complex-valued data from the spectrograms, this illustration however conveys the key idea of regression in the presence of outliers and mixed clusters.
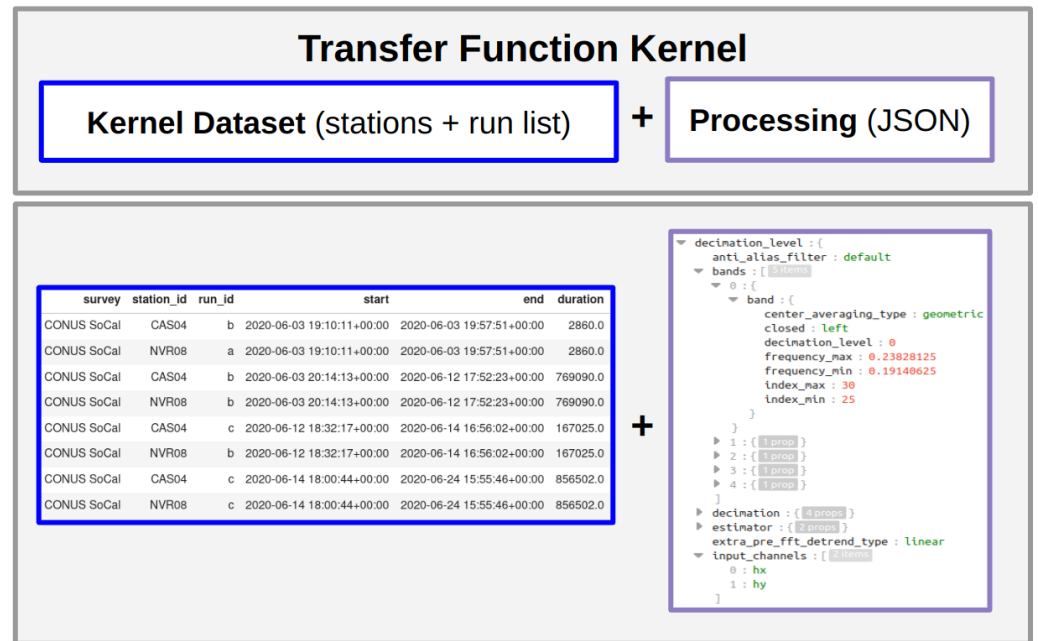
Figure 1: TF Kernel concept diagram: Upper panel represents the TF Kernel with two inlay boxes representing the dataset (Pandas DataFrame) and a processing configuration (JSON). Lower panel illustrates example instances of these structures. Processing configuration image is clipped to show only a few lines.
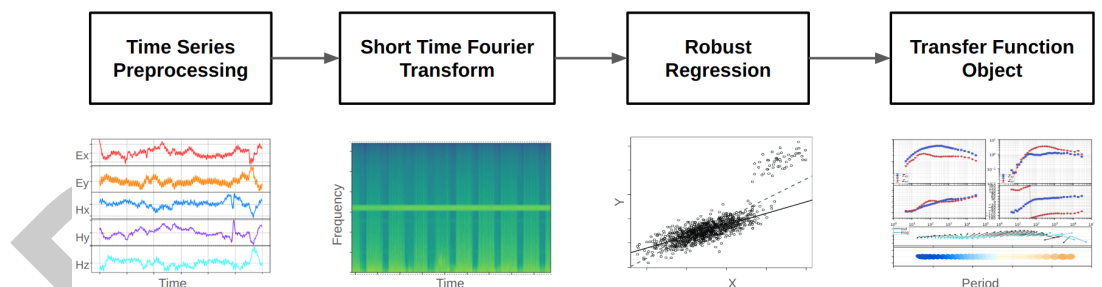

Figure 2: The main interfaces of Aurora TF processing. Example time series from MTH5 archive in the linked notebook (using MTH5 built-in time series plotting), spectrogram from Fourier coeffcient (FC) data structure, regression cartoon from Hand (2018) and TF from SPUD.

Example

This section refers to a Jupyter notebook companion to this paper (archived on GitHub: process_cas04_mulitple_station). The companion notebook builds an MTH5 dataset from the EMscope dataset (Schultz (2010)) and executes data processing – a minimal_example is shown below. Apparent resistivities are plotted in Figure 3 along with the EMTF-generated results hosted at EarthScope EMTF Spud.

```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
```

```
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")
```

92 Code snippet with steps to generate a TF from an MTH5. With MTH5 file
93 ("8P_CAS04_NVR08.h5") in present working directory, a table of available contigu-
94 ous blocks of multichannel time series is generated from RunSummary(). In this example,
95 the file contains data from two stations, "CAS04" and "NVR08" which are accessed from
96 the EarthScope data archives. Then station(s) to process are selected (by inspection of
97 the RunSummary dataframe) to generate a KernelDataset. The KernelDataset identifies
98 simultaneous data at the local and reference site, and generates processing parameters, which
99 can be edited before passing them to process_mth5, and finally export TF to a standard
100 output format, in this case edi.

101 To run the example you must install aurora, which can be done via conda or pip. Detailed
102 instructions and further documentation can be found on the SimPEG (Cockett et al. (2015))
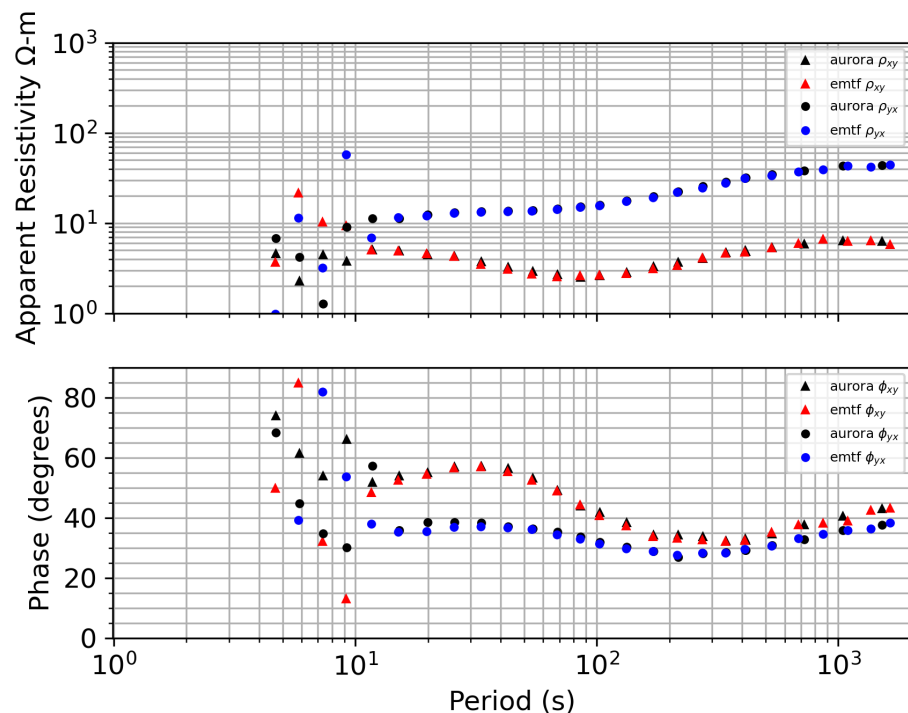103 documentation website.

Figure 3: Comparison of apparent resistivities from Aurora and EMTF for station CAS04. Both curves exhibit scatter in the low signal-to-noise ratio MT "dead band" between 1-10s, but most of estimates are very similar. The Aurora results are from executing the example code snippet. The plotting details are in the Jupyter notebook

Testing

Aurora uses continuous integration (Duvall et al., 2007) via unit and integrated tests, with ongoing improvement of test coverage. Currently, CodeCov measures 77% code coverage (core dependencies mt_metadata and MTH5 at 84% and 60% respectively). Aurora uses a small synthetic MT dataset for integrated tests. On push to GitHub, the synthetic data are processed and the results compared against manually validated values (from Aurora and EMTF results) that are also stored in the repository. Deviation from expected results causes test failures, alerting developers a code change resulted in an unexpected baseline processing result. In the summer of 2023, wide-scale testing on EarthScope data archives was performed indicating that the Aurora TF results are similar to those form the EMTF Fortran codes, in this case for hundreds of real stations rather than a few synthetic ones. Before release to common PyPi and Conda Forge repositories example Jupyter notebooks are also run via GitHub actions to assert functionality.

Software Modifications

Aurora uses GitHub issues to track tasks and planned improvements. We have recently added utilities for using a "Fourier coefficient" (FC) layer in the MTH5. This allows for storage of the time series of Fourier coefficients in the MTH5, so the user can initialize TF processing from the FC layer, rather than the time series layer of the MTH5. Prototype usage of this layer is already in Aurora's tests, but not part of the normal workflow. Noise suppression techniques, for example coherence and polarization sorting and Mahalanobis distance (e.g. Ajithabh & Patro (2023), Platz & Weckmann (2019)) could help reduce noise bias in the transfer functions. A graphical data selection/rejection interface with time series plotting could allow users to manually weight data. The `TransferFunctionKernel` information could be implemented into both the MTH5 and the output EMTF_XML (Kelbert (2020)) for completeness. Unit and integrated tests could be expanded, including a test dataset from audio MT band. There are plans for Aurora to be co-developed with mt_metadata, MTH5 and MTPy-v2 to maintain the ability to provide outputs for inversion and modeling. These improvements would support community participation in a comparative analysis of the open-source 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. Processing is relatively simple and requires very limited domain knowledge in time series analysis. Aurora also serves as a prototype example of how to plug processing into an existing open data and metadata ecosystem (MTH5, mt_metadata, & MTpy-v2). Aurora can be used as an example interface to these packages for the open source MT community, and these tools can contribute to workflows that allow more focus on geoscience analysis, and less on the nuances of data management.

Acknowledgments

References

Ajithabh, K., & Patro, P. K. (2023). SigMT: An open-source python package for magnetotelluric data processing. *Computers & Geosciences*, *171*, 105270. https://doi.org/10.1016/j.cageo.2022.105270

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*. https://doi.org/10.1016/j.cageo.2015.09.015

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. https://doi.org/10.1111/j.1365-246x.1997.tb05663.x

Egbert, G. D. (2002). Processing and interpretation of electromagnetic induction array data. *Surveys in Geophysics*, *23*(2-3), 207–249. https://doi.org/10.1023/A:1015012821040

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. https://doi.org/10.1111/rssa.12315

Hoyer, S., & Hamman, J. (2017). Xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, *5*(1). https://doi.org/10.5334/jors.148

Kelbert, A. (2020). EMTF XML: New data interchange format and conversion tools for electromagnetic transfer functions. *Geophysics*, *85*(1), F1–F17. https://doi.org/10.1190/geo2018-0679.1

Peacock, J., Kappler, K., Heagy, L., Ronan, T., Kelbert, A., & Frassetto, A. (2022). MTH5: An archive and exchangeable data format for magnetotelluric time series data. *Computers & Geosciences*, *162*, 105102. https://doi.org/https://doi.org/10.1016/j.cageo.2022.105102

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. https://doi.org/10.1093/gji/ggz197

Schultz, A. (2010). EMScope: A continental scale magnetotelluric observatory and data discovery resource. *Data Science Journal*, *8*, IGY6–IGY20. https://doi.org/10.2481/dsj.ss_igy-009

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. https://doi.org/10.3389/feart.2020.00296

193 Vozoff, K. (1991). THE MAGNETOTELLURIC METHOD. In *Electromagnetic Methods*
194 *in Applied Geophysics: Volume 2, Application, Parts A and B*. Society of Exploration
195 Geophysicists. https://doi.org/10.1190/1.9781560802686.ch8