**MT Data Processing Software at OSU – Gary Egbert**

***Overview:*** Data processing software used at OSU for routine EarthScope TA processing is based on the Fortran77 EMTF package, together with Matlab programs for plotting, and to provide user interfaces for specific applications. EMTF grew out of programs developed ~35 years ago (Egbert and Booker, 1986) which were modified and extended over the decade following (Egbert, 1997; Eisel and Egbert, 2001). They have been freely available for download, and have been used in some form by a number of research groups around the world. EMTF breaks the processing into two stages: first there is a program (called ***dnff***) which ingests time series for a single run at a one site, along with appropriate meta-data, and converts to frequency domain Fourier coefficients, converted to physical units. Results are stored in FC files, one for each site/run. The second processing step ingests one, or a group, of FC files (for multiple runs/sites) and computes transfer functions (TFs). There are two distinct programs for this in the EMTF package: ***tranmt*** computes single station or remote reference estimates; ***multmtrn*** does “array processing”, combining FC files from two or more sites using the multivariate statistical approach described in Egbert and Booker (1989), and Egbert (1997). Multmtrn can be used for TF estimation, and there are several groups that do use this somewhat routinely. It has more frequently been used (by the author at least) for understanding signal and noise characteristics (Egbert et al., 2000), and for specialized applications, rather than for MT TF estimation. At OSU we generally used tranmt for routine EarthScope processing. Because code is written in F77, all arrays must be allocated at run time. Although the programs are written so that actual array use can be determined at run time, and maximum array sizes are all set with parameters specified in include files, changes in these maximum array sizes require recompiling programs.

The main Matlab programs include some basic tools for plotting apparent resistivity and phase (***apresplt***), a fairly sophisticated and complex GUI-based time series plotting program (***TSplot***), and a GUI that we developed early in the EarthScope program, and used subsequently for routine processing of survey data for each year (***ArrayManager***). There is also a suite of matlab programs (***SS*** and related), developed as an interface for field computers: a program run in matlab prompted users to enter meta-data, ran the Fortran EMTF programs (single station only – hence the name SS), and plotted results, allowing field crews to assess data quality before leaving the site. There are many other related programs (Matlab, Fortran, perl scripts) used for archiving, and specialized applications, that we mostly ignore here. There are also some Matlab classes developed at OSU over the years for specialized data processing applications that might serve as at least initial prototypes for new developments. In the following we give a few more details on the processing codes that are central to basic TF estimation and summarize some of the Matlab codes which might be useful, before suggesting some initial design recommendations for a new community processing code base.

**EMTF components**

***dnff*:** This first step converts from time to frequency domain. Long experience suggests that MT data is usually best processed by dividing the original time series into a series of short sections, windowing, and FT’ing each window. The rationale for this short-window FT approach is two-fold: MT TFs are very smooth in frequency (hence no need for high frequency resolution), and noise is often (but not always!) localized in time. Robust schemes, which rely on rejecting unusual observations, work best if good resolution in time is maintained. Thus, the input of *dnff* is a multi-channel time series () channels for normal MT) of length , and output is a complex array  of dimension  , where  is the number of time segments and  the number of frequencies (controlled by the length of the segment/window). Nominally  , but this is not exact, because segments are usually overlapped (and frequencies near the Nyquist are normally discarded).

The lowest frequency that can be resolved is controlled by window length—nominally this is  where  is the length of the window, but again, in practice this is only approximate. To get lower frequencies we use a “decimation” scheme: after processing the input time series and FT’ing overlapping segments of length , the time series is low-pass filtered and decimated, e.g., by a factor of 4, so that the sampling rate becomes . Then the windowed FT process is repeated, resulting in FC’s at lower frequencies. This process is repeated (for a total of = 3-6 “decimation levels” usually). Thus, the output of *dnff* is actually a series of complex arrays  . In *dnff* all of the parameters that control the decimation, filtering, and windowing, are set in an input file, and can be changed. In practice, for a particular type of data (e.g., NIMS data, with 1 hz sampling) a standard control file is used. In dnff, each time window is demeaned, tapered, pre-whitened and FT’d. System responses (and effects of any internal digitial filtering) are corrected, so that FCs are in physical units (e.g.,  ).

One very important aspect of this scheme is maintaining universal (for the whole survey, at least) control over timing. This is necessary, because in the next processing step we are going to combine FC files from multiple sites (at least two for remote reference), and we need to ensure that windows for different sites correspond to the same time interval, and are identified relative to a fixed reference time, so they can be aligned. To ensure this, a **clock zero** must be defined – this is a date/time, before the start of the first run, where time segment 1 begins. This fixes the times where sets at any decimation level can begin, and provides a numerical tag for each segment. Thus, FCs in two files at two different sites that are in time segment 10234 correspond to exactly the same time window. Of course, it is essential to keep track of segment numbers, zero times, etc. in the output FC files. *Dnff* is fairly general in treating gaps and missing data; provided these are flagged appropriately in input time series, it can handle just about anything, and tries to “make the best of things”, e.g., filling tiny gaps, but omitting sections with too much missing, all dependent on decimation level.

There is a variant on ***dnff*** that has been developed (and only used) at OSU: ***winFT*** a matlab code that does the decimation and filtering in a slightly different way. The advantage of this alternative scheme is that it allows a simple back-transformation to the time domain. We suggest that this scheme (which can easily include the old dnff scheme as a special case) be used as a basis for design of a modern clean processing code. Details will be provided below under design recommendations.

***tranmt:*** This program does the actual transfer function estimation, using FC files produced by dnff as inputs. The MT impedance is estimated for the simple linear model, e.g., for the first row of the impedance tensor

 ,

using a regression M-estimate. The estimates are computed for a series of frequency bands; each band is identified through the decimation level, and a band of frequencies. Thus, for 1 hz data with 128-point windows, one band might be decimation level 1, and FC #s 9-11, with nominal frequency 10/128 hz. The list of bands for which estimates are desired is provided as input to the program. Because we generally want to keep these estimated at the same frequencies for use in inversion, a fixed band file is usually used (i.e., generally no user interaction), but the program is flexible so that any bands can be chosen. Obviously the specification of frequency bands is not independent of the choice of windows used for dnff. Data from any number of runs (each in a separate FC file) can be processed simultaneously, and data from a second site can be used for remote reference. The program reads all files from one or two sites, and for each band collects FCs for all segments, and all frequencies in the estimation band, aligning on segment number. When FCs two sites are input for remote reference processing, only segments available for both are used. The resulting complex data arrays are input to a standard (complex) regression M-estimate, or the generalization for remote reference. Some ad-hoc weighting schemes are also implemented, to reduce leverage, and to further down-weight segments with large residuals in multiple predicted channels. A “coherence weighting” scheme can also be used optionally. Error covariances are computed using asymptotic theory for the regression M-estimate, with minor corrections that are documented in obscure appendices of Egbert and Booker (1986) and Eisel and Egbert (2001). Outputs of *tranmt* are a so-called Z-file. This is non-standard but is more-or-less the basis for the xml TF files documented in Kelbert (2020). TFs are output for any number of dependent (predicted) channels, usually Ex, Ey, and Bz (with Bx and By the predicting channels). Measurement coordinate systems are used for all variables, along with meta-data necessary for transformation to other coordinates. All information required to construct full error covariances, in any coordinate system, are provided (an advantage over EDI). The same output structure/format is used for single station, remote reference, and multi-station processing, so it is easy to consistently merge estimates obtained in different ways. A basic transfer function program, essentially equivalent to *tranmt* would be straightforward to code in a modern way; preliminary, mostly complete matlab versions are available. Basic components/characteristics required for a basic robust TF program are summarized below.

***multmtrn***: I consider this Fortran program obsolete, and I have not personally used this for some years. There is a new object oriented Matlab version (class *ArrayProcessing*), which I am now using extensively for various purposes—but not really for MT processing. I suggest that *multmtrn* not be part of what is developed immediately, but that the other components be developed in a way that allows these capabilities to be easily added at a later date. There is some discussion of developing something based on the Matlab *ArrayProcessing* class for routine processing (by USGS) of new TA MT data. If something useful comes out of this, maybe we could consider developing an additional module to allow use of this approach in the new facility. More broadly, making it easy to incorporate other TF modules as alternatives to a basic *tranmt* module should be a design goal of any new software development. For example, an alternative scheme for robust TF estimation has been implemented by Smirnov (2003), and a module that supports this approach could perhaps be added as an option.

**Matlab Codes**

***TSplot:*** This is a fairly well developed and sophisticated time-series plotting program. It has been around for a long time and is pretty stable. It is GUI based, and has lots of buttons, options, etc. Unfortunately, it was developed mostly in very old versions of matlab and uses features that are at the least deprecated. There have been a few changes to matlab in the past few years that have required minor recoding. If refactored and rewritten even a bit it could be improved. It is possible that other off-the-shelf plotting programs could be used in place of this, but there are also specialized things that are useful (e.g., it is possible to interactively mark bad sections in the time series, and then save a list of bad sections to a file). The program is reasonably stand-alone, but it is called from several of the higher-level user interface programs discussed below. I suspect that this could be used almost as is, but some minor clean up and improvements would make it more useful, and more robust.

***apresplt:*** This is a set of matlab functions and scripts that reads Z-files and make plots of apparent resistivity and phase. In fact, this should perhaps be viewed as an additional (simple) processing step. Outputs of tranmt are in measurement coordintates (not necessarily orthogonal, and not necessarily the same for electric and magnetic) and is only at this stage that everything is transformed into some standard coordinate system where results can be looked at. Although perfectly functional there is not much point in even looking at these codes now – they were written in Matlab 2! All variables are matrices, even triple subscripts did not exist.

***ArrayManager:*** This is a rather complex GUI written to “manage” data processing for the EarthScope array. Basics were written in the first pilot year of the EarthScope project by Egbert, and the code was refined and improved in the next years by Egbert, with some later additions by Kelbert. There are several important and useful features. (1) It lays out visually the times of all sites and runs, graphically showing site overlaps in time. (2) There are buttons created for each run, and for each site, so one can interactively select runs or sites as inputs for various plotting and processing options. (3) Using a standard directory structure, there are scripts which run when *ArrayManager* is started that do all standard processing steps to make sure everything is updated. Thus, for example, if there are new data files found, *dnff* is run to make corresponding FC files. (4) There are buttons and sub-GUIs that streamline processing steps, making everything “point and click”. Thus, one can choose a run by selecting the appropriate run button, and then fire up *TSplot* to examine time series. Choosing overlapping runs from two sites, both can be plotted together to check visually for coherence. Or if runs from two sites are selected, choosing “remote reference processing” opens a new GUI with a suggested configuration for processing. This can be edited to fine tune, before starting *tranmt*. Then a site can be selected to get a list of all processed runs for this site (single station, remote reference, different processing options) any of which can then be plotted using *aprseplt*. *ArrayManager* has been very useful for EarthScope – but in fact it was purpose built to streamline project workflows, as we went. In fact, for long-period data, I guess this program would still work well. For broad-band data with shorter occupation times (1 day instead of 3 weeks), and fewer simultaneous sites (3-6 instead of ~20) some redesign would be helpful. And, this is relatively crude programming in places, developed by multiple academic users over many years, so code maintenance might be a challenge. Starting over on this (in python for example) would probably be too costly, so it might be worth considering if we want to try and make modest changes to improve the matlab code, and to link it to new processing and plotting modules.

***Additional (Object Oriented) Matlab Processing Codes:*** There are some matlab classes that have been used at OSU for specialized processing tasks, that might provide a conceptual starting point (and to some degree prototype) for specification of a new processing system.

These include:

***TS***: a simple class for storing/manipulating time series. Used for merging, decimating, and plotting. Not integrated with much of anything else. Mostly mentioned here because there should be a class for TS in the new system.

***WinFT***: As noted above, this is a class that duplicates some of the features of the F77 code ***dnff***. The main conceptual difference is the way that decimation is handled: time series (TS) are high-pass filtered (HP), to cut off all frequencies below something like  (where, as above, is the window length to be used for FT at the first level). This is tapered, and FT’d as in dnff; keeping track of tapers and windows, it is simple to invert the sequence of FCs to recover the HPTS. The corresponding low-passed time series (LPTS), obtained by subtracting the HPTS from the original TS, is then processed in the same way, with the cut-off frequency reduced, for example, by a factor of 4. Another set of FCs are obtained from this second level HPTS, using time windows that are a factor of 4 longer. The process is continued to obtain, as with ***dnff***, a sequence of FC arrays, one for each decimation level. The factors by which the cut-off frequency is decreased (and time-window length increased) are input variables, along with the HP filter coefficients. The main advantage of this scheme is that the sequence of FC arrays, together with the final LP residual, can be used to reconstruct the original TS. The output is interchangeable with dnff output for TF estimation, but some new processing ideas (that take advantage of a time domain representation) could be readily added. It is suggested that this scheme should be considered as at least an option for the windowed FC used in the new system.

***ArrayProcessing (and related):*** As noted above, a greatly improved variant on ***multmtrn***, described in Smirnov and Egbert (2012). Implementing something like this in the new processing software should probably not be our first priority. The biggest advantage over the Fortran version is that this matlab code allows for missing data. One aspect of the code that should be emulated: it can take as input outputs of multiple FC schemes (dnff, WinFT, and others). Also, there are classes such as *TDataHeader* used to accumulate metadata for arrays of sites. Some of our (Maxim Smirnov, Gary Egbert) work on this might be useful in setting up similar classes for the new system.

**Initial Thoughts on a New Processing System**

It is suggested that existing OSU codes be used as a starting point for design of a new processing system, and that some of the ideas developed as matlab codes also be considered. The following general specifications are suggested for new codes that are developed. Details need to be filled in, but a few ideas are provided as a starting point. In addition, we might want to consider if/how some of the matlab interface codes (discussed above) can be upgraded to provide additional tools to simplify processing, in at least the short term.

1. Open source with a permissive license.
2. Modular, and object oriented, probably using Python
3. Divide processing into three stages: FT and TF estimation, plus post-processing (conversion to fixed coordinate system, plotting, etc.) as in EMTF
4. Define standards for interfaces, inputs and outputs of FT and TF components, develop basic standard implementations, allowing for variants that might be developed later, or contributed by developers in the community.

**FT component**

Inputs:

1. Time series: class? Support input of standard storage formats, plus simple flat ascii. Make it easy to add new subclasses that handle input from alternative time series formats. Best to attach a survey dependent **zero-time** to the TS object from the get-go!
2. Metadata: class? Need to include everything needed to describe the time series. Some properties (start time, sampling rate, system response) are necessary for the FT code; others just need to be carried along with the output FC file (somehow) for later uses.
3. Control parameters: window: lengths, tapers, offsets, FCs stored; decimation: factors, filters, etc. Details might depend on algorithms—design for flexibility (e.g., polymorphism). Probably these should also be designed as classes.

Outputs:

1. Class that contains FCs for all decimation levels (could be just one, so this is a generalization, not a restriction). Enough information to easily reconstruct time series—i.e., start time, zero-time, sample rate, all window and decimation control parameters. All original TS metadata. A standard file format for storage should be implemented and documented (even if processing streams do not always write to disk, should be able to store results).

Components (methods, supporting classes, etc.)

1. System response correction: this would be linked to the metadata class (using system response), but might have to integrate information about other internal digital filtering (FT class implementation dependent) to create a table (response as a function of frequency) that could be used to convert FCs to standard physical units. A key thing here would be to develop this as a stand-alone class that could be used by the FT class. This would be polymorphic, with subclasses easily added to support different specifications of metadata. (Or this step could be kept outside of the FT computation, and only a simple system response table would be allowed as input – but still need this class somewhere!)
2. Decimation/filtering from decimation level n to n+1; controlled by decimation class
3. TF and inverse FT: controlled by windowing class

**TF component**

Inputs:

1. Set of FC objects as output by FT component
2. Estimation band specification
3. Control parameters for estimator
4. Optional list of time segments to use/omit from processing—or more generally prior weights

Outputs:

1. TF class, including full error covariance
2. Optional diagnostics?

Components (methods, supporting classes, etc.)

1. Initialize set of input FC objects for processing task
2. For each frequency band extract appropriate FCs, and organize into simple arrays for TF estimation
3. Regression M-estimate (or alternatives) to estimate TFs for one frequency band
4. Collect results into output FC class

**Post-process and Plot**

Probably this can be viewed as something a TF estimate class does—can store output of TF estimation step in measurement coordinates, transform to a different coordinate system, and make plots. This is pretty much developed in matlab and perhaps also in MTpy. Not much to do here probably.