DOE/ER/14595--T1

Distribution Category UC-60

MULTISTATION MAGNETOTELLURICS

FINAL REPORT
1 January, 1996 - 30 June, 1997

GARY D. EGBERT

College of Oceanography
Oregon State University
Oceanography Admin Bldg 104
Corvallis, OR 97331-5503

PREPARED FOR THE
U.S. DEPARTMENT OF ENERGY
OFFICE OF BASIC ENERGY SCIENCES
UNDER GRANT DE-FG03-96ER14595

DISCLAIMER

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. References herein to any specific commercial product, process, or service by trade name, mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

MASTER

DISCLAIMER

**Portions of this document may be illegible
electronic image products. Images are
produced from the best available original
document.**

1 Project Overview

With previous DOE support (DE-FG06-92ER14277) we have developed the foundations of a practical multivariate approach to processing magnetotelluric "array" data. During the initial funding period an initial version of the processing program, which we refer to as **multmtrn** was written and tested on a several small data sets. These included wide-band MT data contaminated by severe cultural noise (the three site Multistation MT data, which was collected as part of DE-FG06-92ER14277, plus a series of two-station remote reference arrays south of the San Francisco Bay area). DE-FG03-96ER14595 was funded as an extension of this earlier effort to bring this project to a reasonable and orderly termination, by publishing results, and completing a version of **multmtrn** (with appropriate documentation) for release to the community of MT users. This report summarizes these efforts.

Compared to current standards for magnetotelluric (MT) data processing, the multivariate approach that we have developed is unique in that all available data channels are used simultaneously. Here we only outline the approach. Further technical details can be found in Egbert [1997]. In difficult circumstances (e.g., low signal in the dead band from 0.1-10. Hz; cultural noise) the multivariate approach offers significant advantages: signal-to-noise ratios are optimized by using all channels to define the signal, unusually noisy data segments ("outliers") in individual channels are easily cleaned up in an automatic fashion, and statistics diagnostic of coherent noise complications are available. Using **multmtrn** we achieved significant improvements in apparent resistivity and phase estimates in initial tests, even when compared to results obtained from a more standard robust processing code of the sort which is currently considered "state-of-the-art" in the EM induction community (see Egbert [1997] for details). In the course of this project we have gained significantly more experience applying the multivariate processing approach to a wide range of additional data sets. Some examples are given in subsequent sections of this report.

An important aspect of this project (and of the initial development and testing of **multmtrn**), has been the collaboration with a more applied end user, Prof. H.F. Morrison of U.C. Berkeley. Prof. Morrison was formally involved as a co-Principal Investigator on DE-FG06-92ER14277, and has continued an informal collaboration in this project. This collaboration has helped to keep the research directed toward problems of practical significance, and has helped to provide the impetus and the means to test ideas on realistic MT survey data. In the course of this project **multmtrn** was set up on computers run by the UCB Engineering Geoscience group at Lawrence Berkeley Laboratories. The program was used, in consultation with the OSU PI Egbert, for processing data for several projects. Selected results from these projects are described below.

2 The Multivariate Approach to MT Data Processing: A Brief Summary

We sketch here the multivariate statistical approach we have developed for robust processing of multiple (two or more) station MT data. Further details and initial examples

are provided in Egbert [1997]. Some additional examples and discussion of results are presented in the next section.

To allow for the possibility of coherent noise we have adopted a multi-stage procedure. In the first stage the MT array data is decomposed into coherent and incoherent components, outliers in all channels are cleaned up, and an appropriate scaling (i.e., the incoherent noise amplitude) is determined for each channel. This is accomplished for channel j by: (a) robustly computing principal components (PCs) of (some or all of) the remaining data channels; and (b) estimating the incoherent noise scale from the residuals to a robust regression of channel j on all significant PCs. The procedure is iterated, with cleaned up data (from the previous iteration) used to compute the PCs for subsequent iterations, and residual variances are converted to approximately unbiased estimates of incoherent noise variances by solving a small system of linear equations (see Appendix A, Egbert [1997]).

Using the cleaned scaled data, issues of coherent noise and optimal impedance estimation can be addressed. As a first step in this stage we compute diagnostics for coherent noise. Useful diagnostics include eigenvalues of the scaled spectral density matrix, canonical coherences, canonical covariances, and eigenvector plots. these diagnostics allow us to determine which stations and which frequency ranges (if any) are contaminated by coherent noise. If we find that coherent noise is not a serious problem, impedances can now be estimated. For this purpose we propose several robust variants on the multivariate errors-in-variables estimator, which we refer to generically as RMEV estimators. These schemes use information from all data channels to define optimal reference fields, and to protect against outliers in all components.

When MT data is severely contaminated by coherent noise (as for the example small arrays south of San Francisco, CA. considered in Egbert [1997]), we have not found any completely general or automatic "statistical trick" which can guarantee reasonable results. However, the multivariate approach allows a greatly enhanced understanding of signal and noise characteristics, which can (in some circumstances) allow us to greatly reduce biases due to coherent noise. One general approach we have found useful is to compute diagnostics for coherent noise contamination which are resolved in both time and frequency. To stabilize (and reduce random errors in) these diagnostics (which have only a small number of degrees of freedom) we first project the data vectors onto the coherent signal/noise space. Time/frequency resolved diagnostics can clarify the degree to which MT parameter estimates are likely to be contaminated by coherent noise, and allow us to identify which parts of the data are least effected.

Provided coherent noise is intermittent, an initial data screening based on these diagnostics can greatly improve MT parameter estimates. Furthermore, once we have reasonable "starting guess" estimates of MT transfer functions, the geometrical character of coherent noise fields can be estimated, and robust schemes can be tailored to eliminate or downweight such noise throughout the data, further reducing bias.

In summary, there are three (possible) goals of multivariate analysis with **multmtrn**. The first is to understand the character of the signal and noise. Can the situation be reasonably characterized by the standard quasi-uniform (plane wave) MT source assumption with incoherent noise added to each channel? Or is there evidence for consistent coherent noise? The program produces output which make it fairly simple to verify if the former

case holds, and to gain some understanding of the nature of coherent noise. The program also produces output which can be used to generate "generalized transfer functions" for more complicated (e.g., non-plane wave) sources. The second goal is to get the best possible plane wave MT transfer functions (this is really the ultimate goal—but in a sense it really follows logically after the first goal). For this purpose one has to make some decisions about what best defines the MT (plane wave) signal. The program will follow default rules (i.e., the strongest signal overall), but this can lead to very poor results. Often the user might know that one or more sites are (relatively) unaffected by coherent noise cultural noise, while other sites are seriously contaminated. In a case like this the user can provide information to the program which can significantly improve the final results. This is not something the program can do automatically. The user must provide this information to the program (through command line options and control files). A final use of **multmtrn** is to closely scrutinize badly contaminated data to try and pull out usable results. The program can help the user to identify the "bad" data (time segments and/or stations) in some cases. There are command line options in **multmtrn** which activate some of these features. These remain experimental, and will probably be useful in certain specialized situations. We give an example below of one such case. However, it is not expected that many users will find these experimental features useful, unless significant further effort is put into development of better user interfaces and explanatory documents.

3 Some Example Applications

One major activity supported by DE-FG06-92ER14277 has been application of **multmtrn** to processing of a diverse set of MT data sets. Through this process we have gained more insight into the advantages and limitations of the multivariate approach. Application to additional data sets has lead to simplifications in the programs use, and significant generalization to allow for more general experimental configurations. Because several of these projects have been collaborations with others, this activity has allowed us to test ease of use and documentation. We summarize results from several test applications of **multmtrn** here.

3.1 Carrizo Plain and Parkfield EM Profiling Data

Beyond the initial test data sets discussed in Egbert [1997], the first application of **multmtrn** was to a pair of continuous magnetotelluric profiles across the San Andreas Fault (SAF) near Parkfield, California and near the Carrizo Plain. In Figure 1 we plot apparent resistivity and phase pseudo-sections from the Parkfield profile which crosses the SAF near Middle Mountain [Unsworth et al., 1997]. In this experiment 10 data channels (8 electric, two magnetic) were sampled simultaneously. Here TE mode (E-field parallel to strike) results obtained with standard robust processing code (left panels) are compared to results obtained with MULTMT (right panels). Due to an equipment malfunction no remote site was available for referencing. Results for both processing methods are thus "single station". In the left panels, a band of severe downward bias is evident in the

apparent resistivity pseudo-section near 0.1 hz. This an example of the well known downward bias problem, caused by (autocorrelated) noise in the local reference magnetic field channels. Near 0.1 hz signal-to-noise ratios (SNRs) in the magnetic fields are often very low due to low levels of natural signal, and to high levels of cultural and ground motion noise. Note that day-to-day variations in SNR are evident in the clustering of downward bias amplitudes in groups of three dipoles - the number of TE mode dipoles collected in a single days run.

This narrow-band bias in apparent resistivities is essentially eliminated by the multiple station processing with **multmtrn**, which uses all 10 data channels simultaneously. Note that frequency independent static shifts - a sort of "geologic noise" which no statistical processing can eliminate - are still evident between dipoles for both processing schemes. The multiple station scheme succeeds in removing the autocorrelated noise bias because **multmtrn** automatically determines noise levels in all data channels, and then uses optimal linear combinations to define the reference fields. The behavior of the MULTMT estimates is consistent with theory, which shows that approximately unbiased estimates should result even with "single station" multiple dipole EMAP (but not necessarily with 4 or 5 channel MT) data. This illustrates another advantage of the multiple station approach-with enough local data channels it can readily be used to obtain at least approximately unbiased estimates even when a remote site is unavailable for referencing. Note also that **multmtrn** does a better job of cleaning up noise in some of the lower frequency estimates (e.g., amplitude and phase for the four lowest frequencies beyond 3500 m).

The geologic implications of this survey turn out to be quite exciting. Inversion of the cleaned up data reveals a narrow (500 meter wide) zone of high conductivity beneath the surface trace of the SAF beneath Middle Mountain. [Unsworth et al., 1997],interpret this as evidence for high concentrations of saline pore fluids in the upper 4-5 kilometers of the fault zone. The EM profiling data also locate the contact between very resistive Salinian granite basement rocks on the west side of the SAF, and more conductive Franciscan assemblage rocks to the east, with an accuracy of a few hundred meters. The location of this deep contact is of obvious importance to siting of a possible deep drill hole into the SAF at this location. Both of these results depend critically on the TE mode data plotted in Figure 2. This data was only useful after removal of bias with **multmtrn** was successful.

Selected pseudo-sections from a second continuous tensor EM profile across the SAF at the Carrizo Plain are presented in Figure 2. In this experiment 15 data channels (8 electric, two magnetic in the EMAP profile, plus 5 channels at a remote reference site) were sampled simultaneously. During the experiment the remote site was moved once. Since one set of EMAP dipoles was run with both remotes, it was still possible to reference all E and H field components across the profile to a common "normal field". This allowed us to estimate total electric and magnetic fields which would be observed across the profile for a pair of orthogonal source polarizations. In contrast to more standard methods, this sort of processing is straightforward with the multivariate approach.

Plotted on the left are TE mode magnetic field pseudo-sections - i.e., estimates of the real and imaginary parts of the magnetic fields which would be produced along the profile by electrical currents flowing parallel to the SAF. At each frequency the real part of the

magnetic fields averages to 1.0, and the imaginary to 0.0. Deviations from these averages are indicative of lateral structure. The real part is simplest to interpret in a qualitative fashion. Areas of relatively enhanced current flow along strike (higher conductivity) have real parts of H exceeding 1.0 (purples); relatively resistive areas have real parts less than 1.0 (yellows and reds). The surface trace of the SAF is near 4000 m; the purple "blob" near 1 hz shows a zone of enhanced conductivity in the near surface fault zone. The variations of H at lower frequencies are indicative of the contrast in deep basement conductivity between the two sides of the fault. Of course this sort of pseudo-section can be formally inverted as part of a more quantitative interpretation.

Similar pseudo-sections have been produced for other source polarizations or modes, and for other field components. As a further example, in the right hand panels we plot apparent resistivity and phase pseudo-sections computed from total electric fields across the profile corresponding to the TM mode. For this mode electric currents flow perpendicular to the SAF. Static shifts of apparent resistivity are evident where the surface trace of the fault is crossed. This shift is probably mostly due to topographic effects as the EMAP profile crosses the very steep exposed fault scarp. The total electric and magnetic fields for each source polarization are currently being used by Prof. M. Zhdanov (Univ. of Utah) for experiments with a novel EM migration scheme.

multmtrn proved its value in processing this EM profiling data set in several ways: (1) by providing a natural framework for calculating the full array transfer functions (TFs) needed for estimation of total field components needed for the migration scheme. (2) by providing a simple and optimal way to eliminate bias effects in the local/remote horizontal field TFs needed to construct the magnetic field pseudo-sections. (3) by helping to identify significant cultural noise at some frequencies at Carrizo Plain (apparently due to activity in nearby oil fields). MULTMT allowed us to detect this problem and eliminate severely contaminated frequencies from further analysis.

3.2 Sea Floor MT from the Gulf of Mexico

As part of our testing of **multmtrn** a version of the program was ported to run on computers at Lawrence Berkeley Laboratories, where the program was used by graduate students in the Engineering Geoscience program at UC Berkeley. Multiple station processing was applied to data from two experimental seafloor MT surveys in the Gulf of Mexico [Constable et al., 1998; Hoversten et al., 1998], and to an experimental natural source induced polarization (IP) survey conducted near Battle Mountain, Nevada. For both experiments results from the multivariate processing were compared to those obtained with the default MT processing software available with the commercial EMI MT-1 system, and with the standard robust remote reference (RRR) processing programs of Egbert and Booker [1986]. A detailed discussion of the results of comparisons for the Gulf of Mexico seafloor survey are given in the Masters thesis by Scott [1997].

The goal of the MT survey in the Gulf of Mexico was to demonstrate the feasibility of reconnaissance mapping of the base of buried salt domes using seafloor MT data. This information, which can be of great value in assessing possible value of oil prospects in the Gulf, is difficult to obtain with seismic techniques. In this survey data were collected with 1-2 land reference sites, 1-2 seafloor magnetometers and several (2-3 or more) electrome-

ters running simultaneously. There are thus multiple sites running at all times, but with variable and changing station configurations. The datasets are thus well suited to multivariate processing. In general these datasets were noisy, with a number of instrumental and procedural problems being worked out as the experiment proceeded.

In general, Scott [1997] concluded that the multiple station program produced significantly improved results, compared to conventional and RRR estimates. Examples of the comparison, taken from Scott [1997], are presented in Figure 3.

Comparison on the Battle Mountain IP survey, for which data quality was very high, did not reveal any significant improvements from application of a multiple station approach. The difference between the two cases illustrates a point of some generality: When data quality is high, the multiple station approach does not generally improve results. The computationally intensive multivariate approach is thus only justified in cases where some problems with data quality are evident. At present, run times for **multmtrn** are great enough that it makes sense to first process with a much faster conventional RRR scheme, and reprocess with the multiple station approach only when this is necessary, or if some of the more specialized outputs of **multmtrn** are required.

3.3 MT Survey in a Culturally Noisy Area of Bavaria

A fourth area of application has been to long period MT data from Germany near the German Continental Deep Drilling (KTB) site. This dataset consists of 25 MT stations in a profile across the western part of the Bohemian Massif. The survey was conducted in a collaborative project between the GeoForschungsZentrum Potsdam and the Geophysical institute of the Czech Academy of Sciences. Work on this data set at OSU has been a collaboration with Dr. Markus Eisel, a German post-doc from GeoForschungsZentrum Potsdam. Dr. Eisel has been visiting OSU for the past year, in part to work on multiple station processing methods. This work has focused on more advanced applications of the multiple station processing scheme, including further development of diagnostic statistics for choosing time segments of minimal cultural noise contamination. The new KTB survey data was collected in small arrays of 5–6 stations, so this dataset is again ideal for experiments with multiple station processing.

Data from this survey have proven to be very noisy, with apparent resistivity and phase curves estimated using conventional processing schemes exhibiting non-physical behavior similar to that seen in the SF Bay Area MT sites discussed in Egbert [1997]. In Figure 4 we plot apparent resistivity and phase curves for an example MT site (SPA) processed using three different approaches. First, on the left are results from robust remote reference processing using an MT site 20 km west of SPA as the reference. In this figure there are very clear non-physical biases in the MT parameter estimates, particularly between 1–10 seconds. In the center, we plot results from multiple station processing using data from 6 simultaneously recording sites and the magnetic fields of the reference station to define the reference field (see Egbert [1997] for a description of this approach). Although the non-physical biases are almost completely removed by this approach, there is still a suggestion of small residual biases in one mode of the impedance. On the right we plot results from applying processing as in the center, after first using diagnostics from an initial multiple station processing to reject data segments identified as being extremely

contaminated with coherent noise. This additional step eliminates the rather suspicious hump in the apparent resistivity curve near 1 second period.

The sorts of biases seen in the German dataset are in fact fairly typical of MT data taken in many parts of Europe, where cultural noise contamination is often severe. A multiple station approach may thus be exceptionally valuable in such areas.

3.4 Parkfield/Hollister Earth Quake Monitoring Array

A second project conducted in collaboration with researchers at UC Berkeley involved application of multiple station processing methods to a small array set up to test for possible precursory electromagnetic signals before earthquakes. For the past two years, the Engineering Geoscience group at U.C. Berkeley and the Berkeley Seismographic Station have run an experimental EM monitoring array consisting of two five component MT stations—one near Parkfield, California and the other approximately 150 km north near Hollister, California. Data are collected continuously at both sites at 40 hz and 1 hz. We have worked with graduate students in Engineering Geoscience to apply multiple station processing methods for routine and experimental processing of this unique data set.

This dataset has highlighted one of the major advantages of the robust multiple station approach embodied in **multmtrn**: noise and outliers are explicitly allowed for in all channels. With a conventional robust remote reference (RRR) approach, outliers in the remote reference channels can significantly degrade MT parameter estimates. For the UCB EM monitoring array, there were significant sporadic sources of very large amplitude noise at both sites. These problems were especially severe at the Hollister site (SAO). Thus, when SAO was used as a remote reference for the Parkfield site RRR estimates of MT impedances were frequently useless. Results obtained for four days with both RRR and **multmtrn** processing are given in Figure 5. For three of the 4 days, the RRR results are very noisy, while the **multmtrn** results are uniformly quite good. The high incidence of bad days (3 out of 4) for the period selected for this plot was a bit extreme—typically only for 1 day out of 5 did the RRR estimates produce such bad results. Furthermore it would be possible to edit out the worst sections for the bad days and significantly improve the RRR results. However, for this project with continuously collected data there was a significant advantage to having an automated processing approach which essentially always worked.

The value of allowing for, and then cleaning up, outliers in the remote reference channels is further illustrated in Figure 6. Here we display robust single station, remote reference and multiple station results from an EM profiling survey conducted by University of Washington and Oregon State University in the fall of 1997. The single station results are very smooth, but exhibit a clear bias in both apparent resistivity and phase curves near a period of 7 seconds. Using a remote reference makes the results much noisier, clearly showing that the remote site is too heavily contaminated by noise to be useful. Using **multmtrn** produces the best results: the curves are again smooth, and now nearly free of the biases seen with single station processing.

4 Documentation and Dissemination of **multmtrn**

In the process of working with other researchers in applying **multmtrn** to the diverse data sets discussed briefly above, this program has been significantly improved. Many bugs have been fixed, and a number of new features have been added to make use of the program easier and more practical for a wider range of real world applications. The new version of the code is available to interested MT researchers via MTNet

<http://www.cg.NRCan.gc.ca/mtnet/mtnet.html>

or by contacting the PI (egbert@oce.orst.edu).

In addition to a formal publication on the theory (Egbert, 1997; attached), we have also prepared a technical document describing installation and use of **multmtrn**. A copy of this program documentation, is included as Appendix A to this report. During this project we have also updated documentation for our previously released robust single station and remote reference processing programs. For completeness copies of this updated documentation is included as Appendix B.

In the course of this project we have also developed a set of matlab M-files which provide means to plot results and to further explore the diagnostics output by the multiple station programs. A document outlining these codes is provided as Appendix C.

Finally, we are currently collaborating with Electromagnetics Instruments (EMI) to make versions of our processing programs (including some form of **multmtrn** available to commercial users as an integral part of the software that will be delivered with the new generation MT-24 system. This work is currently being pursued by Dr. Markus Eisel, with financial support from EMI.

5 References

- Constable, S., Orange A., Hoversten, G.M., and Morrison, H.F., Marine magnetotellurics for petroleum exploration, Part I; *Geophysics*, in press, 1998.
- Egbert, G. D. and Booker, J. R., Robust estimation of geomagnetic transfer functions, *Geophys. J. R. astr. Soc.*, **87**, 173-194.
- Egbert, G. D., Robust multiple station magnetotelluric data processing, *Geophys. J. Int.*, **130**, 475-496, 1997.
- Hoversten, G.M., Morrison, H.F., and Constable, S., Marine magnetotellurics for petroleum exploration, Part II; *Geophysics*, in press, 1998.
- Scott, C. L., The application of robust and robust multivariate processing to experimental magnetotelluric data sets, M.S. Thesis, University of California, Berkeley, 100 pp, 1997.
- Unsworth, M. J., P.E. Malin, G.D. Egbert and J. R. Booker, Internal structure of the San Andreas Fault at Parkfield, California *Geology*, **25**, 359-362, 1997.

Figure 1 : Results of processing Parkfield TE mode EM profiling survey data with two different approaches. Left: Single station robust processing. Right: robust multiple station processing. Note that for this data no remote site was available, so standard remote reference processing was not an option. Note also that there are 10 simultaneous channels available for each setup. The multiple station approach eliminates downward bias evident in the single station data around 01. hz.s

Figure 2 : Selected results from multiple station processing of Carrizo Plain EM profiling data. Left: anomalous horizontal magnetic fields for TE mode. In this mode electric currents flow parallel to the strike of the San Andreas Fault. Areas of enhanced H (purples) correspond to concentrations of currents, and hence higher conductivities. Right: TM mode electric Field pseudosection. In this mode electric currents flow perpendicular to the fault strike.

Figure 3: Comparison of results from processing seafloor MT sites with robust remote reference (top two panels) and robust multiple station approach (bottom two panels)

Figure 4 : Results from three different processing approaches applied to MT site SPA from near the KTB drilling site. Left, Robust remote reference processing using a remote site approximately 20 km west of SPA. Center: Multiple station processing using data from 6 simultaneously recording sites. Although the non-physical behaviour near 1-10 seconds periods are significantly reduced, there is still evidence for some bias in this band. Right: Same as the center, but now diagnostics from an initial multiple station run are used to reject data segments identified as most severely contaminated.

Figure 5 : Estimates of apparent resistivity and phase obtained for four consecutive days (Julian days 64-67, 1996) from the two station UC Berkeley EM monitoring array in Parkfield and Hollister, CA. Above are estimates obtained with standard robust remote reference, below are multiple station estimates.

Figure 6 : Results from processing EM profiling data from the Parkfield 1997 survey three different ways. Eight channels of E-field data were collected in the profiling setup, along with a 5 channel remote MT site. Here we plot a representative MT impedance from setup 101. Left: robust single station results are smooth, but there is a spectacular example of downward bias due to noise in the local H fields near a period of 7 seconds. The noise in H at these periods probably results from ground motion. Center: remote reference estimates are very noisy due to a very noisy remote site. Right: Multiple station processing cleans up outliers at the remote site, and yields estimates which are smooth and nearly free of obvious bias effects.

Appendix A:
Documentation for MULTMTRN:
A Program for Multiple Station
Analysis of Magnetotelluric Data

MULTMTRN: A Program for Multiple Station Analysis of Magnetotelluric Data

Gary D. Egbert*

February 27, 1998

1 Overview

This document describes use of the robust multiple station transfer function program described in Egbert, 1997 (Robust multiple station magnetotelluric data processing; GJI). The program is an extension of the robust single station and remote reference processing programs based on Egbert and Booker, 1986. Before running **multmtrn**, time series files must be processed by program **dnff** to convert to frequency domain Fourier Coefficient (FC) files. Note that these are the same FC files used by the single station/remote reference processing program **tranmt**. These files contain a series of FCs obtained from Fourier transforming a series of short overlapping time segments for each of a series of decimation levels. By careful use of **dnff**, short time windows from different stations which overlap in time will be coincident, and tagged with consistent *set numbers*. **multmtrn** uses these set numbers to align coincident sets. For details on how to use **dnff** (and **tranmt**), see documentation for **EMTF**.

To run the program a file (called **array.cfg** by default) is required. This file is roughly analogous to the **tranmt.cfg** file used by **tranmt**. This file tells how many "stations" (i.e., groupings of channels, each of which is contained in a separate file (or set of files)), names of FC files, output file root, and some preliminary "weights", which can be used as one way to emphasize certain channels in the definition of plane wave sources. In addition, there are a large number of command line options which control functioning of the program. These are described in greater detail below. Here we summarize the key points and features.

There are three (possible) goals of multivariate analysis with **multmtrn**. The first is to understand the character of the signal and noise. Can the situation be reasonably characterized by the standard quasi-uniform (plane wave) MT source assumption with incoherent noise added to each channel? Or is there evidence for consistent coherent noise? The program produces output which make it fairly simple to verify if the former case holds, and to gain some understanding of the nature of coherent noise. The program also produces output which can be used to generate "generalized transfer functions" for

---

*College of Oceanic and Atmospheric Sciences, Oregon State University, Corvallis; egbert@oce.orst.edu

more complicated (e.g., non-plane wave) sources. The second goal is to get the best possible plane wave MT transfer functions (this is really the ultimate goal—but in a sense it really follows logically after the first goal). For this purpose one has to make some decisions about what best defines the MT (plane wave) signal. The program will follow default rules (i.e., the strongest signal overall), but this can lead to very poor results. Often the user might know that one or more sites are (relatively) unaffected by coherent noise cultural noise, while other sites are seriously contaminated. In a case like this the user can provide information to the program which can significantly improve the final results. This is not something the program can do automatically. The user must provide this information through command line options and control files. A final use of **multmtrn** is to closely scrutinize badly contaminated data to try and pull out usable results. The program can help the user to identify the "bad" data (time segments and/or stations) in some cases. There are command line options which activate some of these features. These are very experimental, and are probably useful in certain specialized situations. It is not expected that many users will find these experimental features useful.

The program is reasonably flexible about the numbers and types of channels in each station (i.e., FC file). However, output formats for some files are really sensible only for more-or-less conventional 3-5 channel MV or MT sites. However, the program works with, and has been extensively used with, EM profiling data with multiple E_x and E_y channels, and even with non-EM data. It is also possible to have **E** and **H** channels in separate FC files. Note however: **multmtrn** breaks channels into *groups* for two purposes: (1) to estimate levels of incoherent noise all channels from different groups are used to predict the channels of a fixed group; (2) to estimate *local* transfer functions (e.g., impedances). For both of these purposes it is simplest to use the default channel groupings defined by the FC files. Furthermore, for some purposes it is only possible to group channels which are contiguous (i.e., next to each other in the ordering implied by the FC files and the order of FC files in the **array.cfg** file. Finally, for some purposes, the program expects that local magnetic channels H_x and H_y should be the first two channels in any station grouping. Thus, if possible follow the conventions established in **EMTF** and make sure that the horizontal magnetic channels occur first in the FC files.

In principal, the program works for one station ... but this only makes sense if there are multiple E or H setups (e.g., an EM profiling data set with 10 channels—e.g., 2 orthogonal H and 4 setups of orthogonal E.) Even in this case there may be problems (suggestion: run with the option **-GA**).

The program produces a variety of output files, some of which have the same format as the outputs produced by **tranmt**. Roughly these are of three sorts (1) files which characterize the full array response, giving all inter-station and inter-component transfer functions, plus signal and noise characteristics for all channels. These files are either ASCII files with explanatory headers, or there are matlab scripts to open and read these files, and to extract and plot the most commonly useful things. (2) so called Z-files which give local transfer functions, plus all information needed to calculate error bars in any coordinate system. In general there is one of these files produced for each station grouping with more than two channels, which are the predicting channels under the assumption of quasi-uniform MT sources. By default station groupings are determined by the FC files (one FC file = one group), but this can be changed with a command line option. The

format of these files is identical to that of Z-files produced by single station or remote reference analysis conducted with **tranmt**. Again there are matlab M-files for reading and plotting Z-files. See also the document *Z-Files*. (3) output files containing intermediate or auxiliary results which have been useful during development of the programs (and which may be useful for some advanced/experimental applications. These are in general poorly documented. None of these files are produced by default, and many of the command line options are to produce these files. Note also that several of the files output are redundant and will be eliminated in future releases. The input control file and the most significant output files are described further below.

A final warning: This program is complicated—in many places more complicated than necessary. It will be impossible to understand many of the options for **multmtrn** without a thorough understanding of Egbert (1997). Note also that significant computer resources might be required to run **multmtrn**, particularly for large data files, or if there are many stations/channels. The program is still under development (very slowly due to funding limitations), and is only partially documented. Some options are not described anywhere, and obviously not all combinations of options have been tested, or even make sense. There are almost certainly bugs which will show up when the code is used by others. Please notify egbert@oce.orst.edu if you run into problems. I stress again that used blindly, this program is quite capable of yielding quite dreadful results. User beware.

2 Making the Executable

There is a simple Unix Makefile for compiling and installing **multmtrn**. Before making the program there are several things to be aware of. First, maximum number of data channels (for one station), and maximum number of station need to be set in the header file **nstamx.h**. Most of the parameters that need to be changed most commonly are in this include file. Here is a brief description of the meaning of the parameters:

- *nstamx* Maximum number of *station*. Note that a station consists of all channels grouped together in a FC file. Note that there could be several FC files (same group of channels, but a different run) for one station.
- *nchmx* Maximum number of channels for a single station grouping
- *ngrpms* Maximum number of groups for local TF computations. Normally this could be the same as *nstamx*, since each station is a natural (and the default) group for local TF estimates. But this might not always be true.
- *nchemx* Maximum number of predicted channels for a single station. Normally this would be *nchmx* - 2, but if the channel groupings used for local TF computations are modified by use of the *-s* option, this could change. E.g., with one set of **H** channels, and multiple sets of **E** channels, one could estimate all impedances in a single file, all predicted by the same H_x , H_y . In this case one could have *nchmx* = 3, but *nchemx* would have to be at least as large as the total number of **E** channels.
- *nbmax* Maximum number of frequency bands to compute estimates for. Actually this seldom needs to be changed.

There are other parameters which might have to be modified in `iosize.h` if the size of FC files (typical number of decimation levels, time segment lengths, etc.) are changed. If you use the FC program “as is” you generally should not need to change parameters in this file; otherwise some changes to this might be needed. Note that this file is basically the same as the parameter include file used for `tranmt` (of the same name). Any parameters in the include files not explicitly discussed here should be left set as they are.

- *ntfmax* Maximum number of data points to allow for for a single frequency band. Increase for very long time series. It also might be necessary to decrease the size of this to get the program to fit into memory, particularly in parameters like *nstamx* and *nchmx* are large.
- *ndmax* Maximum number of decimation levels to allow for. Should be set to whatever is used in `dnff`.
- *nfreqmax* Maximum number of FCs saved for a single decimation level. For example, with 128 point sets this would be at most 64, but is often set to less. Again, this should be set as in `dnff`.
- *ntpmax* Maximum number of FC files for a single station. Each FC file would correspond to a different “run” of the same set of data channels. Often this could be set to 1.
- *lpack* This is a logical parameter set to *.true.* when the FC files are stored in packed integer format. In this format one complex number is stored in 4 bytes. Set this the same way it is set in `dnff`. Packed format is the normal usage. However, for data recorded with 24 bits resolution, you might want to use standard binary format (i.e., set *lpack* = *.false.* in both `dnff` and here.
- *lfop* This is a logical parameter normally set to *.false.* Setting this to *.true.* makes the program open and close all FC files before and after every read. This is necessary on some systems when the number of stations in the array is too large, since some systems limit the total number of files that may be opened in a Fortran program.

A second point to note in making the executable is that the program links to the `lapack` library. Source code for this public domain library can be obtained from

<http://www.netlib.org>.

For completeness I have included the necessary routines in `../lapack`, including the full set of “blas”, or basic linear algebra subroutines. If you don’t have lapack on your system, go into the source directories `blas_src` and `qr_src`, and make `libblas.a` and `libqr.a`. Then in this directory (MMT), edit the Makefile as indicated (in the Makefile comments) to make paths to the necessary libraries correct (if lapack is on your system, you might still need to edit the library path.) With all of this done, just type *make multmtrn*.

3 The array configuration file array.cfg

Here is an annotated array control file for "multmtrn". This file tells which data files to use, and also controls relative weighting of normalized data vector components for determining the reference fields for plane wave (quasi-uniform) source transfer functions.

```
2                                <=== # of stations in the array
../CF/bs_nod.cfg                <=== band set up file (as for tranmt)
1 10                            First station; number of FC files, # of ch.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1.  inverse weights for each channel
../FC/f10spa121.ts4            - FC file (one for each file for station #1)
S12                             - Site name (used in some output file names)
1 5                             Second station; number of FC files, # of ch.
1. 1. 1. 1. 1.                 as for station # 1
../FC/f5remw11.ts4
R02                             - Site name (end of station 2)
A12TS4                          <=== Array name ... used for output files
                                <===OPTIONAL :
                                <===OPTIONAL :
                                <===OPTIONAL :
                                <===OPTIONAL :
```

Note on optional arguments: As soon as the EOF is reached, the program stops looking for options. To specify the second optional argument, you also provide the first, etc. The optional arguments are:

- θ , rotation angle for x -axis for output in `array_name.cfg` file
- maximum number of iterations for robust iterations (obsolete)
- A series of lines to control set numbers to be included in analysis; The first line gives the number of lines ND to follow. For each of the following ND lines the range of set numbers to use for one decimation level is given as: id, iband(1,id), iband(2,id)

Note on weighting for each component: set inverse weighting to zero to omit a component from definition of reference (this is equivalent to assigning a very large inverse weight)

4 Output Files

Following is a brief description of the *standard* output files produced by multmtrn ... additional output files are generated when certain command line options are used. These are discussed separately. Also, there are MATLAB routines for reading all of these files. This allows for plotting, merging results from different processing bands/sites, reformatting for input into other programs, or whatever other manipulations might interest the user. This matlab interface is currently only in the initial stages of development, but if you're at all

familiar with matlab, this will be the easiest way to interact with these output files. See the separate matlab documentation and the help facility in the EMTF matlab directory for more details. (Note: the matlab functions and scripts use other functions and scripts from various of the subdirectories (matlab/IN, matlab/MISC, etc.). It will be easiest to put all of these subdirectories (i.e., matlab/* in the matlab search path (defined by the environment variable MATLABPATH, or by the matlab command "path").

Note that there are two general classes of output files: those which refer to array results (things which make sense for the whole set of stations together) and those which refer to a single station (e.g., MT impedance estimates). The latter sort of files are output for EACH appropriate station in the array. Where possible these have the same format as that used for output from "tranmt". This makes it relatively easy to merge results from array processing, with more conventional robust transfer function results.

Output files for array results all have a common root, with the file type identified by a 1-3 character suffix (e.g., *.M , *.TER , *.S0). Note that in early versions of this code the file type was put at the beginning of the file name (e.g., M_* , TERR_* , S0*). This change has been made for greater consistency with PC file naming conventions.

Note: There is an advanced feature of the code which allows for processing of "sub-arrays" ... that is, portions of the data which are available for only a subset of stations. We will not discuss this (only partially debugged) feature here, but note that when this feature is invoked, additional "sub-array" output files will be produced. For completeness we note these here. I cannot guarantee any functionality of any aspect of this advanced feature.

4.1 Standard array output files

Note that the file root **array_name** used in the example output file name examples given here will be replaced by whatever is given at the end of the **array.cfg** file (i.e., in the last required line of this file; see above).

4.1.1 array_name.M

This ASCII file gives eigenvalues for each period band, along with the vectors which span the coherence space (i.e., the columns of W in Egbert (1997). Output is in a tabular form, with numerous explanatory headers. Results are given in a separate table for each period. All "significant" eigenvectors are output; the number of vectors per band is variable. See the example file for further explanations. This file might be useful for an initial look, but is in an awkward format for further manipulation/plotting. Note that before outputting the columns of W, H components are scaled to be of order one, and E components are converted to the same units as H (nT) by assuming a 100 ohm m apparent resistivity (i.e., for plane wave sources over with an apparent resistivity of 100, E components will also be of order 1; for $\rho = 10^{*4}$, E components will be of order 10, etc.). For vectors which do not have much plane wave content, E could be much bigger than H, and things could get ugly. Also, note that the output format is really only reasonable for "standard" arrays. In particular it was designed for MV arrays with many 3 channel sites. with the scaling described above, it also works OK for multiple 5 channel MT arrays. However, for

for more general arrays table headers are wrong, and some entries may fail to fit into the allowed format (so you'll get lots of *****). Formats can be changed by editing subroutine `prteig.f`.

4.1.2 `array_name.S0`

This is a "RAW" BINARY SDM file which contains all information needed to specify the array (station locations and IDs, orientations and IDs for each data channel, etc.), together with estimates of incoherent noise variances, and the SDM of the cleaned data. This file can be read by the matlab scripts `IN/sdm_init` and `IN/sdm_in`. Note that when the sub-array feature is used results for sub-arrays will be output in files named `array_name.S1`, `array_name.S2` etc. (Not well tested; probably does not work). Also note: In previous releases `array_name.S0` files contained the leading eigenvectors/values instead of the full SDM. These, along with canonical coherences/covariances, eigenvalues for subsets of channels, etc. are easily computed in matlab. An interactive/GUI set of scripts for analysis of the SDM in matlab using results output in this file. (If you have matlab: look at/try `sdm_plot.m`) This file is now the main full array output file.

4.1.3 `array_name.SN`

This ASCII files gives a frequency domain summary of all signal and noise spectra, including eigenvalues (but excluding eigenvectors). Useful for plotting signal and noise spectra; could be eliminated, since this info is also in `array_name.S0`. This file can be read into matlab with the command `sn_in('array_name.SN')`, and plots can be made in matlab using `sdm_plot.m`. See the EMTF matlab document for more details.

4.1.4 `array_name.Pw`

This binary file contains the full estimated "array TF" for an assumed plane wave source. All info necessary for computing TFs relative to any chosen pair of reference channels (along with error bars) is given in the file. Read with matlab routines `Pw_hd` and `Pw_in`. Matlab routine `Pw_plot` can be used to rotate and plot components of array TFs with error bars. The command line option `-L` suppresses this output.

4.2 Individual station files

4.2.1 Z-files

These files contain estimates of local transfer function. They are identical in form to the Z-files output by `tranmt`. By default they contain transfer function estimates relating the last `nch - 2` channels at each site to the first 2 channels at that site. File names are determined from an input root specified in the `array.cfg` file, and a station identifier. Local TF files computed by `multmtrn` have the prefix `.zmm`. For a standard 5 channel MT station, the first two channels are the horizontal magnetic channels, so these files contain vertical field TFs, along with the impedance tensors. For an EMAP site all E-field TFs are given. In all cases everything is output in the MEASUREMENT coordinate system,

along with all necessary orientations. For each frequency band two complex Hermitian matrices are also output (one 2x2 matrix specifying the "input signal covariance", plus a $nch - 2 \times nch - 2$ matrix specifying the residual covariance for the predicted channels). Together these matrices can be used to compute error bars for components expressed in any rotated coordinate system. This file can be read into matlab with routine `Z.in`. Apparent apparent resistivities and phases can be calculated using matlab routine `apresplt`. See "*Matlab M-files for EMTF and multmtrn*" for details.

Also: see `Z_files.ps`, a postscript document file containing a more detailed description of the Z-file format

Note: With the option `-s` (see below under options) it is possible to change the default grouping of channels by stations, so that channels in separate FC files can be combined into a single group. This allows, for instance, for H fields in one file to be used as the "local reference" fields for one or more sets of E fields present in separate file(s).

Note: These error matrices are computed differently for different processing schemes (e.g., single station, Remote reference, multiple station), but formulae used to convert these matrices to error bars are identical for all schemes. Z-files can thus be merged across processing schemes.

Note: in the example here there are two Z-files produced, one for a 10 channel EMAP site (`S12_A12TS4.zmm`) and the other for a 5 channel MT site which was installed as a remote (`R02_A12TS4.zmm`).

4.3 Specialized output files

Output of these files is triggered by a command line option. These specialized/experimental files are described briefly along with the corresponding command line options below.

5 Running multmtrn

By default information about the array (number and location of data files, names for output files, some program control options) are found in the file `array.cfg` (e.g., see the example above). The name of this main program control file, and a number of processing and output features can be changed with the following command line options. Note that the following only provides a summary, using terminology and ideas developed in detail in Egbert (1997; "Robust Multiple Station Magnetotelluric Data Processing, GJI, in press"). Understanding this paper is more-or-less a prerequisite for understanding many of the options. Options marked with an asterisk (*) are unlikely to be of interest to most users, and are more for debugging/testing. Many of these options are only partially (often barely) tested, and are too hard for me to try explaining at this point. Most are probably not worth pursuing, and will probably disappear from future releases. Among these latter sort of options, not all possible combinations have ever been tried or even make sense.

6 Command Line Arguments

There are a number of command line options which either modify the way the estimates are computed, or modify outputs from the program.

6.1 General purpose command line arguments

--

"multmtrn --" generates a summary of usage and command line options

-f[array_file]

change default array file name to **array_file**

6.2 Arguments which control estimation options

-n

turn off all robust features (runs MUCH faster, but sometimes you get what you pay for)

-m

turn off robust RMEV (downweighting of outliers in individual data channels), but still do robust (rotationally invariant) SDM stack)

-i

change default max # of iterations for robust regression for each inner loop estimate of local noise; Default is set in main program as *itmax_ln_d*

-o

Change default # of outer loops for local noise estimation/individual channel outlier cleanup; Default is set in main program as *itmax_cln_d*

-I

change default max # of iterations of iterations used for robust regression for final TF estimate; Default is set in main program as *itmax_rrr_d*

-T

turn ON automatic timing error correction In this case a file called **array_name.TER** is sought; if this is found timing offsets are read from the file for each station and used to phase shift appropriate channels. If the file is not found, the program estimates the timing corrections and writes a file of this name. If times are all correct, the timing shifts should be zero. With default usage, multmtrn tries to open this file to find any needed timing shifts (in seconds).

If the file does not exist, the program estimates timing shifts by looking at the variation of phase with frequency of inter station magnetic field TFs near the Nyquist frequency. The estimated timing shifts are then output in this file, so any further runs for this array can skip the timing error estimation step. This file also provides a way for the user to intervene and manually adjust the timing correction. Note that timing corrections are now turned off by default; use the `-T` option to turn this on.

`-N`

Don't transform eigenvectors in `.M` output file (default is to output linear combinations of the first two eigenvectors for which the magnetic fields are, on average, polarized N-S and E-W).

`-G[option]`

change channel grouping for coherent noise variance estimation. option should be one of:

`T` $\Rightarrow$ all components of a type at a single site (default)

`S` $\Rightarrow$ all components at a single site

`A` $\Rightarrow$ each component by itself

`-g *`

try to sort out plane wave/gradient sources geometrically (NOT FOR SMALL (# of stations OR spatial extent) ARRAYS!!!!)

`-R#`

Use projection of magnetic fields from station `#` into coherent signal/noise space to define plane wave reference.

`-s[file_name]`

change default definition of channel groupings for individual station TF output. By default channels are grouped by "stations", with all channels in a single FC file are assumed to be a single station. For each such station with at least 3 channels an individual station TF file (named `STA_ARRAY.zmm`) is output. With this option the groupings of channels to use for these Z-files are given in a file, denoted here as `[file_name]`. Here is an example used for an array in which the E and H channels were in separate FC files. There were 3 H setups (2 with 2 channels, 1 with 3), and 3 2 channel E setups, for a total of 6 FC files. The `array.cfg` file for this array looked like this:

```
6
../bs_nod
1 2
```

```

1. 1.
  ../.../FC/H1.f2
S1H
1 2
1 1
  ../.../FC/E1.f2
S1E
1 3
1. 1. 1.
  ../.../FC/H2.f3
S2H
1 2
1. 1.
  ../.../FC/E2.f2
S2E
1 2
1. 1.
  ../.../FC/H3.f2
S3H
1 2
1 1
  ../.../FC/E3.f2
S3E
TEST3

```

To have S1H and S1E together treated as a station (and S2H S2E, etc.), run multmtrn with the option -scgrp.cfg where file **grp.cfg** contains the following text:

```

3           <===== # of channel groupings
S1          <===== name for first group
4           <===== number of channels in first group
1 2 3 4     <===== channel numbers; following order in array.cfg
S2          <===== name for second group ... etc.
5
5 6 7 8 9   <===== channel numbers for group two; files f3H2, f2E2
S3
4
10 11 12 13

```

To always use the two H channels at the first site for a reference, and to omit the Hz TF at site 2 **grp.cfg** would be changed to:

```

3           <===== # of channel groupings
S1          <===== name for first group

```



```

4          <===== number of channels in first group
1 2 3 4    <===== channel numbers; following order in array.cfg
S2         <===== name for second group, etc.
4
1 2 8 9    <===== channel numbers, group two; for files f3H2, f2E2
S3
4
1 2 12 13

```

Note in particular that channels may be used in multiple groups, or omitted.

-M[SNR_min]

Only use sets with signal to noise ratio (as determined from 2 dominant eigenvectors) exceeding input argument SNR_min

6.3 Arguments which add to or change output files

-L

Don't output **array_name.Pw** file. By default this file is output, but it is often not used for routine processing.

-Z

Rotate channels into a common coordinate system before outputting in **array_name.M** file. Coordinate rotations require channels to be paired. When this option is invoked the program tries to make reasonable assumptions about channel pairings. In particular it assumes that an H_x is always followed by an H_y , and an E_x by an E_y . If this isn't the case (e.g., in an EMAP type configuration) things will be screwed up.

-c[option]

where options is one of H, R, or N ... output canonical coherence file. The file will be named **array_name.CC**. The options essentially control normalization of channels...are as follows:

R = canonical coherence, the default if only -c is specified, normalizes each data channel by the square root of its total variance.

N = canonical covariance, normalizes each channel the estimated incoherent noise scale

H = canonical covariance also, but with H channels expressed in nT, and E channels scaled to nT using a crude estimate of the array average impedance

The output file has the format:

⇒ ASCII

⇒ one header record with two integers : total number of channels (all stations, number of frequency bands)

⇒ followed by one more "record" for each frequency band : period, canonical coherences (or canonical covariances) for each station relative to all other stations. In general there are as many canonical coherences at each site as there are data channels. Thus, with 3 5 channel sites, the first 5 channels are coherences of site 1 relative to sites 2&3, the next 5 are for site 2 relative to 1&3, etc. Up to 15 coherences are plotted on each line. Multiple lines are used if the total number of constituents exceeds 15. If there are more components at one site than there are at all other sites combined (e.g., 2 sites, one with 10 the other with 5 channels, as with remote reference EM profiling) some coherences will be zero.

-C

output correlation matrices for data, and for all incoherent noise groupings. (For incoherent noise groupings, the correlation matrix is estimated from the correlation among residuals for all channels in each group, as predicted by all channels in all other groups. Note that the correction applied to variances to make these nearly diagonal is not allowed for in this calculation).

The output file has the format:

⇒ ASCII file ... with lost of "informative" headers ... self explanatory?

* -P

Output principal component (PC) "time series" file **array_name.PC**. The frequency domain array data vectors bfX_t can be expanded as a linear combination of the significant PCs:

$$X_t = \sum a_{jt} W_j \quad (1)$$

The PC file contains the coefficients a_{jt} , which can be used to reconstruct the coherent part of the Fourier coefficients for all channels and time segments.

The output file has the format:

⇒ Binary ; see output routines in **pc_out.f**.

* -t

Output transfer function files used in local noise variance estimation

* -r

Output the **array_name.PC** file AND output the raw array data vectors bfX_i (in a single file, still called **array_name.PC**).

The output file has the format:

⇒ Binary ; see output routines in **pc_out.f**.

6.4 Arguments for coherent noise downweighting

All this stuff is very experimental, and will only be useful in some special cases, e.g., as discussed in Egbert, 1997.)

-a *

omit data sets specified at end of array.cfg file only on first coherent noise downweighting iteration; on subsequent iterations coherent noise/signal ratios are used to decide which data should be omitted (This is a way to use the coherent noise downweighting feature when a time window of "clean" data can be specified a priori. On the first only data sets in a specified time range are used for initial processing; based on initial results, a preliminary separation of the total coherent signal into coherent noise and MT signal is made. Based on this separation weights are determined for all data sets, and a refined estimate is computed.

-u *

assume signal and coherent noise are uncorrelated for computing weights

-w# *

change default number of iterations for coherent noise downweighting to #

-p# *

change default cutoff for coherent noise downweighting to #

7 Problems

The ASCII output file **array_name.M** is a good place to look to see if everything is working. In particular inter station magnetic TFs should be close to one. If everything looks garbled, and you're sure the data is pretty good, timing errors are a likely problem. Double check that timing is OK, first by plotting. If this checks out, run with the -T option. This should correct small timing errors, which can be hard to see in the time series even when they are significant enough to mess up inter station TFs significantly. Also: make sure channel orientations are correct in FC file headers!

If there are problems with run time errors, first make sure the parameters in nstamx.h are set correctly. My experience is that this is the most common cause of run time errors (and sometimes of garbled results, even when execution terminates normally).

the -C option output correlation matrices ... so you can check the simple coherence between any pair of channels.)

**Appendix B : Documentation for EMTF:
Robust Transfer Function Estimation
for Single Station and Remote Reference
Magnetotelluric Data**

EMTF: Programs for Robust Single Station and Remote Reference Analysis of Magnetotelluric Data : UNIX Version

Gary D. Egbert & Markus Eisel*

March 2, 1998

1 Overview of EMTF

Use of the transfer function (TF) estimation programs in **EMTF** can be broken down into three (or two) distinct steps. These are

- **Reformat The Data (or not!):** In general data files come in lots of different formats, so making a general package of processing software which accommodates all possibilities is difficult. Initially we adopted an unusual, but standard, space efficient format for time series files. Once the data is in this format, the second Fourier Transform (FT) stage **dnff** can be easily run. Translating programs to convert two data formats into this standard format are included with **EMTF**. These are **rfemi** for the EMI MT-1 system, and **rfasc** for simple ASCII files. Translating programs for other input formats can be patterned after these, as discussed further below.

For several special formats **dnff** can be run directly on the raw data files without any translation. These include a simple ASCII format, the EMI MT-24 format, and mini-SEED files. Direct reading of simple ASCII files is supported in the standard version of **dnff**. The more special versions of **dnff** for reading MT-24 or mini-SEED files are not currently part of the standard **EMTF** package, but are available separately. In the long run adapting **dnff** to your particular data format is the best solution for efficient routine processing. The implementation of direct reading of simple ASCII files is a good example of how to do this, as discussed further below. Since no one wants to throw away the original data files, and since you have to cope with modifications to the reformatting program to interface with your data format anyway, it probably makes more sense to skip the reformatting step, and adapt **dnff** to work directly with format of your data files.

---

*College of Oceanic and Atmospheric Sciences, Oregon State University, Corvallis

Note that in early versions of **EMTF** the reformatting step was done by a program called **clean**, which also cleaned up isolated outliers in the time domain. This program is still provided in subdirectory **C**, but it is not currently supported, because it seldom seems to make much difference.

- **Window and Fourier Transform Data Segments:** This is accomplished by program **dnff**. Input to this program is a data file in either the "standard binary" or simple ASCII forms, a file describing system parameters, several files which control program options and (optionally) instructions about sections of data to omit from further processing. Output is a file consisting of Fourier coefficients (FCs) ordered by frequency (i.e. all Fourier coefficients for a fixed frequency are stored together in the file). Key features of **dnff** include a built-in decimation scheme, and a number of features for aligning time windows from a number of stations (to allow for remote reference and multiple station processing of sites collected independently, with possibly different start times and/or sampling rates). The output FC files are used by all subsequent frequency domain processing programs, including single station and remote reference transfer function program **tranmt** and the multiple station program **multmtrn**. As noted above specialized versions of **dnff** designed to work directly with other specific data file formats (EMI MT-24; Mini-SEED) have been developed, and are available from egbert@oce.orst.edu.
- **Robust Estimation of Transfer Functions :** This is accomplished by program **tranmt** (or the multiple station program **multmtrn**; this program is described in a separate document). Input is the ordered Fourier coefficient file(s), and files controlling program options. The program can handle an arbitrary number of data channels (e.g. GDS, MT, EM profiling with multiple in line dipoles, remote reference MT or EM profiling). Fourier coefficients from local electric and magnetic and remote reference channels may be in the same or different files. Output is a file containing TFs and full error covariances, from which all standard MT and GDS interpretation parameters can be computed in any specified coordinate system.

1.1 Directory Structure

Code for each of the steps is in a separate sub-directory. Each has a README file which contains some useful information on compiling and using the programs. There is also a test data directory containing two ASCII files each with 5 channel synthetic data, plus all configuration files needed to execute all processing steps on this data (for single station or remote reference). Sub-directory **test** also provides an example of how the directory structure might reasonably be set up for processing of MT data with **EMTF**. Here is a summary of the directory structure which you will get when you unpack and install **EMTF**.

RF/ ==> Data reformatting programs

RFEMI => Translate EMI MT-1 to standard binary format

RFASC ==> Translate sample ASCII to standard binary format

BIN2ASC ==> Translate a standard binary data file to simple ASCII format

FCOR ==> Compute a look-up table of filter/system response corrections from a standard system parameter (.sp) file (also requires sensors files)^M

C/ ==> Data cleaning (despiking) program

D/ ==> Decimation/FFT program dnff

T/ ==> Single station and remote reference robust TF code

MMT/ ==> Multiple station processing program

include/ ==> include files common to programs in several directories

matlab/ ==> some matlab plotting and post-processing programs.
These are described more fully in the separate document matlab_doc.ps

test/ ==> Some simple (artificial) example data for testing

DATA/ ==> Contains two compressed simple ASCII files test1.asc.Z and test2.asc.Z, plus an example of a clock file .

SP/ ==> Example system parameter files

CF/ ==> Example Configuration File directory, with all files needed for test runs

FC/ ==> empty directory where Fourier coefficients would be put. and following the given instructions

MT ==> empty directory for output from tranmt

sensors ==> (Empty) directory where system parameter look up tables (if any) should be placed (or replace this with a link called "sensors" to the location where such system parameter tables are stored).

results ==> what should appear in your D, FC, and MT directories as each step is completed.

bin ==> empty directory where executables will be installed for testing

doc/ ==> Documentation for programs

TEX ==> latex source files for documentation

PS ==> postscript document files

1.2 File naming conventions

To make consistency with PC versions simpler, file types are indicated by suffixes like .sp, .bin, .cfg, etc. There are two general file types. Configuration files which are used for many runs, and files which all correspond to a fixed station/run. Configuration files all have a suffix of .cfg. Files corresponding to a fixed station/run will generally all have the same root, with different suffixes used to denote binary data files, FC files, system parameter files and result files. In the following examples we use the generic station/run id `test1`, as in the `test` directory. Note that in previous versions file types were denoted by the first few letters in the file name, rather than with suffixes. There are probably still places in documentation or README files where the files are referred to by the old naming conventions.

1.3 Include files

For all programs there are a large number of parameters which can be set to control things like the maximum numbers of data channels allowed, time series lengths to allow for, block sizes used for data files, etc. These are now all set in include files, which are denoted by the suffix .inc. In some cases include files are used by several programs (e.g., parameters controlling block sizes of binary data files are used by the reformatting programs which write the files, and by the Fourier transform program which reads the files.) These include files are kept in the **include** subdirectory to make it easier to maintain consistency between programs. There are also include files which are only used by a single program. These are in the directory with the rest of the source code for that program. Further details on parameters that can be set in the include files are given below under the section for the appropriate program. Include files in **include** are described with the first relevant program.

1.4 Making and testing source code

The best way to get an overview of how to use the programs is to go through the simple synthetic test case in `test`. The directory structure and configuration files in `test` needed for the test runs are all set up. In fact, you will find that the directory structure in `test` provides a template for a reasonable setup for processing real data, and that most of the default configuration files provided will work for most applications. You thus do not need to understand all of the details described in this document to start actually using the

code. Going through the test cases will also help ensure that the code works properly on your system.

In each section below there is a short sub-section giving instructions for running the simple test cases for each program. Rather than reading the whole document through it is probably best to start by skimming through each section and then making the code and following the testing instructions in each of these "Testing" sub-sections. Note that the testing instructions are summarized in the README file in **test**. Beginning from the simple ASCII data files **test1.asc** and **test2.asc** in **test/DATA**, and following the instructions in the "Testing" subsections below you should end up with MT impedances consistent with a 100 ohm-m half space. To check your results, compare to the final and intermediate results provided in the **results** directory.

All of the source code sub-directories have UNIX makefiles. Use **make** to make the executable target (e.g., "**make dnff**"), then "install" the executables for testing by typing "**make install**" (in each sub-directory after making the executable). This will move executables into **test/bin**, a convenient place for testing. Note that for a more permanent installation you will probably want to put the executables somewhere else (and change BIN_DIR in all of the makefiles).

2 Reformatting data

Some programs for translating a few kinds of data files can be found in subdirectories **EMTF/RF/RFEMI** and **EMTF/TR/RFASC**. The first of these directories contains code for a program which converts EMI MT-1 data files into the standard EMTF time series format, while the second converts ASCII data files. We describe **rfasc**. **rfasc** should be viewed as a simple example of more general data reformatting programs. The idea is that by changing a few input routines the translating routines could be readily adapted to other data file formats. Note that the current version of **dnff** can directly read data files in simple ASCII format that **rfasc** takes as input, so this version of **rfasc** is essentially obsolete (unless you want to use the much more compact binary format for storing the data files).

The **rfemi** program is somewhat more complicated than **rfasc**, because **rfemi** also reads the MT-1 data file header and sets up the system parameter file needed for running **dnff**, and because the EMI MT-1 data format is, shall we say, unusual.

For **rfasc** a "clock" file must be supplied to tell the program

- (1) The digitization rate (in SECONDS, not HZ!!!! - i.e. ΔT).
- (2) The clock time for the first sample in the data file.
- (3) The reference, or zero time. All data samples are numbered from this reference time, so that data from simultaneous stations can be aligned. This file can have any name; you will be prompted for the name by the program. (But for consistency with reading of ASCII files in **dnff** where an identical file is required we call this file **test1.clk**).

Here is an annotated example of this file:

```
16.0          <=== sampling rate, in seconds
85 07 20 0 0 0 <=== clock reset: yr,mo,day,hr,min,sec
85 07 20 0 0 0 <=== universal clock zero: yr,mo,day,hr,min,sec
```

Note that the clock reset corresponds to the start time of the data. For a file with no sample numbers this would be the time of the first record in the file. For a file with each sample numbered, this would correspond to the time of sample number zero. The "universal clock zero" corresponds to a fixed time which can be used to define the zero time for all stations which might be processed together--i.e., a time *before* (or coincident with) the clock reset times for all stations which might be processed together.

In addition to the "clock file" you of course must provide an ASCII data file containing the data to reformat. As it is currently set up the program expects the file to contain integers in fixed format, with all *NCH* channels on one line. The program assumes that there are no data gaps.

2.1 Making rfasc

There is a simple UNIX makefile; just type "make rfasc" to make the executable. There are four include files which you might want to edit. In `../..//include` there is a file called **datasz.inc** which controls the size of the data blocks in the standard binary output files produced by **rfasc**. Parameters in this file are used by programs which read the binary data files (i.e., **dnff**). If this file is changed all reformatting programs and **dnff** (plus any other programs which read the data files) should be recompiled. There is also a file in `../..//include` called **four_byte.inc** which needs to be modified to run the programs on a DEC system where record lengths in fortran direct access open statements are specified in four byte integer words, instead of in bytes as on most systems. (Programs have been run on a DEC system; but no guarantees here!) There are also two include files in the source code directory **RF/rfasc** called **nch.inc** and **format.inc**. These are used to set the number of channels and the format of the data file to read. Both of these have to be changed before making the executable for a particular reformatting task. Type **make install** to move the executable into the test bin directory. Change **BIN_DIR** in **Makefile** to install elsewhere.

2.2 Testing rfasc

From **test/DATA** run `../bin/rfasc`. You will be prompted for a "station id", output file name, input file name and clock name. The Station ID is a 3 character string which will be written into the data file for use in subsequent processing and plotting programs. There is an option to concatenate additional data files into the same output file. Answer "y" to add additional input files, and you will be prompted for file names. Note that each file requires its own clock file giving time of the first sample in the file (and the *same*

universal clock zero!) After terminating a single output file, the program asks if you want to continue reformatting files. If you answer "y" here, the program starts again asking for station id, etc.

Here are prompts from the program (marked with >>>) and how you should respond for the first test file:

```
>>> station id
TS1
>>> enter output file name
test1.bin
>>> enter header (80 character max)
testing 1, 2, 3, ...
>>> input file name
test1.asc
>>> enter clock reset file name
test1.clk
>>> another input file? (append to current output file)
n
>>> continue?
n
```

Reformatting of **test2.asc** into **test2.bin** is analogous.

To further test that this worked, you could use **RF/BIN2ASC/bin2asc** to translate the binary file back to an ASCII file. Use of this program is self explanatory.

2.3 Accommodating Other File Formats

By changing the input routines (all in file **inpu_asc.f**) the reformatting program **rfasc** can be adapted to other data formats (But it probably makes more sense do directly adapt **dnff** to your input format.) It should be possible to accommodate almost any file format by changing this file only, by providing a new set of input routines with the same names, arguments, and functions. There are three subroutines in *inpu_asc.f*. Their functions are to

- (1) **Initialize input** [**ininit** (*inunit*)] : this routine should get the name of the file to process, open it, get the start time of the input data (if necessary; some data files may have a time channel). There is one argument :: *inunit*, which is the unit number to be used for connection of the input data file.
- (2) **Position file at start of data** [**frstdat** (*inunit*)] : This routine positions the input file at the start of the data. (i.e., rewind file and skip past any header blocks). For a direct access file, this routine could reset the next sample number to read. Might not be needed, depending on how the general input routine is coded; but provide a dummy routine with this name in any event. There is one argument :: *inunit*, which is the unit number to be used for connection of the input data file.

- (3) **Read a block of data** [*indo* (*inunit,ix1,n,ngot,ipoint,nchp1,lend*)]: Tries to return a block of n of the next data points. *ngot* is the number of points actually returned (to accommodate what happens when the EOF is reached). The routine should return *nchp1* (= # of channels + 1) channels - the first is a "time channel", containing the sample number (reckoned in units of ΔT relative to the universal clock zero). The data is returned in integer array *ix1*. Argument *ipoint* tells where in the array to put the first sample read in the current call to *indo*. See the source code of the current version for details; the main program keeps track of and updates *ipoint* between calls. *lend* is .true. when the EOF is reached, .false. otherwise.

2.4 EMTF Standard Time Series File Format

The output of the program consists of a data file written out as a direct access binary file. The format is simple (but admittedly non-standard), and reasonably compact since only a bit over 2 bytes are required for each number. With this format even 24 or 32 bit EM data can be stored in roughly 16 bits with very little or no loss of dynamic range. The data is written out in a series of blocks—one header block followed by a series of data blocks. Each data block consists of *nblk* data samples (each of *nch* channels), plus a header. Currently *nblk* is set in the include file *datz.inc* in the directory **include**. The same include file is used for making the cleaning program, and the FFT program for the next stage. Data in the output file is stored as 2-byte integers. The header record for a data block contains $2+2*nch$ 4-byte integers. These are, in order: (1) The sample number for the first data point in the block; sample numbers for all subsequent data points in the block are consecutive. (2) The number of samples in the block. If the block is full this will be equal to *nblk*. Data blocks will be full except at the end of a file or when there are gaps in the data. Note that the blocks are written as fixed length direct access records, so all blocks are the same length. (3) the next $2*nch$ integers are (a) an offset and (b) a scale factor for each channel; this gives a greater range to the possible data values allowed. In general for data recorded with 16 bits, the scale factors will always be 1, but there are circumstances when this generalization is useful. Output routines choose the scale and offset automatically to make sure that there is no overflow.

The first block in the output file is a header block of the same length (of course). The first part of the header is written in ASCII characters. The remainder contains information on the number and length of "data segments" (individual segments are separated by data gaps) and approximate scales for each channel. These could be used by a plotting routine but are otherwise unimportant. The information in the header is mostly self explanatory; see routine **wrhdbin** in file **C/out_bin.f**.

As with the input routines, the output routines (in **C/out_bin.f**) are easy to change, but then the corresponding input routines for the next step would also have to be changed.

A note on concatenation of multiple input data files into a single output file: Because the next step (FFT) will assume that the system parameters (including gains, filter settings etc.) are the same throughout the data file, this feature should only be used if all combined data files have common system parameter settings. The program will ask for a clock reset file for each input file. The output file will keep track of gaps between the end of one file

and the beginning of the next.

3 Cleaning of Isolated Outliers and Data File Reformatting

This is accomplished by program **clean** in directory **/C** . As noted above this program is not really supported any more. The version of this program provided is set up for cleaning and reformatting integer data in an ASCII file. The program expects 5 channels of data in (5i7) format (Hx,Hy,Hx,Ex,Ey). To apply the program to data with a different number of channels, change parameter *NCH* in **clean.f** (along with the appropriate data statements). All samples are assumed to be contiguous (no gaps). Use of the program is essentially identical to **rfasc** , and the output format is the same.

4 Windowing and Fourier Transforming Data Segments

The windowing and FFTing of data segments to produce the complex frequency domain data vectors for each station is accomplished by program **dnff**. This portion of the transfer function programs is much more general than is needed for most purposes. The programs have been designed for Fourier transforming time series prior to array processing. To accommodate a range of instruments, sampling rates, etc., a lot of generality has been built into this program. This leads to complicated configuration files which have lots of seemingly useless parameters. If a parameter in a configuration file doesn't make much sense, its probably safest to leave it set as it is in the examples in the **test** directory (or in the other examples discussed here).

The input data file expected by this program should be either in the standard form output by the cleaning program - i.e. direct access binary files with two or four byte integer data, or the simple ASCII file format described below. To read data in the ASCII format use the command line option **-a** (or **-A**; see below for details). (Also see below about adapting the program for other input formats, and note that this has been done for EMI MT-24 and Mini-SEED formats.) Also note that a four byte integer variant on the standard two byte format is supported. The four byte files have the same format, but all two byte integers in the header and data blocks are replaced by four byte integers.

In addition there are (up to) four configuration files needed for a run. In summary these are:

- (1) **decset.cfg** controls the windowing and decimation. This file is required.
- (2) **pwset.cfg** is used to control pre-whitening options. This file is required.
- (3) **test1.sp** is used to specify system parameters. This file is required.

- (4) **test1.bad** tells the programs about "bad data". This file is optional. (Here **test1** is the data file root; this file is always constructed by adding the **.bad** suffix.)

The Fourier transforming scheme is a bit of a mixture between cascade decimation and a standard FFT. A brief explanation (and justification) is given in Egbert and Booker (JGRAS, 1986). The basic goal is to use data segments that are as short as possible, given the desired resolution in the frequency domain. We thus use short (e.g., 128 points) overlapping segments of the input time series to get Fourier coefficients for the highest possible frequencies. We refer to these data sets as "decimation level 1". To get lower frequencies a longer time window is needed. To do this efficiently, the program digitally low pass filters the input series and then decimates the smoothed time series. Short segments (again 128 points, say) of the resulting time series are then windowed and FFTd. These are decimation level 2 sets. This filtering and decimation process is repeated as often as desired (to produce levels 1, 2, 3, 4 ...).

4.1 Making dnff

There is a simple UNIX makefile; just type "make dnff" to make the executable. There are three include files which might need to be edited. In **../include** there is the file called **datasz.inc** (described above) which controls the size of data blocks in input binary data files. This file will not generally need to be changed for any reason. There are two files in the source directory **D**: **params1.inc** and **params2.inc**. These files will have to be changed if you want to significantly change windowing or decimation options set in the **decset.cfg** file, but for most users, parameters in these files can be left alone.

4.1.1 params1.inc

These are general parameters used to define the size of arrays to allocate in the main program **dnff.f**.

numx

Maximum length of windows to be FFTd (in samples)

nsmax

Maximum number of sets to allow for (this is the sum over all sets in all decimation levels)

nbadm

Maximum number of bad record segments

nbytes

Default storage format for binary time series. Depending on NBYTES the the assumed format of the input binary data file is changed.

For *nbytes* = 4, integer*4 is assumed, while for *nbytes* = 2, integer*2 is assumed.

The user can change the storage format using the command line option -b(2or4)

lpack

logical parameter to control packing of FCs in file. If *lpack* = .true. each complex FC packed into a 4 byte integer.

4.1.2 params2.inc

These are parameters which control size of arrays used for decimation and related functions. This file is included in **decimate.inc**, which is included in **dcimte.f**, **decset.f**, **dnff.f**, **fcorsu.f**, **mk_offst.f**, **mkset.f**, **pterst.f**. **params2.inc** is also included in **getsp.f**.

ndmx

Maximum number of decimation levels to allow for.

nchmx

Maximum number of channels to be fft'd

nfcmx

Maximum number of filter coefficients for decimation filter (should be about equal to the maximum decimation factor)

nfilmax

Maximum number of filters to correct for for a single data channel.

4.2 Testing dnff

After making **dnff** and installing the executable in **test/bin**, the program can be tested. Go into the **test** directory. All necessary configuration files are either in the main **test** directory, or in subdirectories **CF/test** and **test/SP**. Look at **paths.cfg**, **CF/decset/cfg**, **CF/pwset.cfg** and **SP/test1.cfg**. Note that the system parameter file **SP/test1.cfg** is very simple (no filters, completely flat system response). To run the program in the default (binary input file) mode type **bin/dnff**, and enter the binary file name (**test1.bin** or **test2.bin**; the files output by **rfasc**) when prompted. To test the program in ASCII mode type **bin/dnff -a**, and enter the ASCII file name (**test1.asc** or **test2.asc**) when prompted. After the run completes, you will be asked if you want to process another file. Answer "n". You will find output Fourier Coefficient files called **test1.f5** and/or **test2.f5** in **test/FC**.

4.3 Configuration and system parameter files

Here are some further details on the configuration files (which control windowing, decimation, etc. options for **dnff**), and the system parameter files (which specify corrections for system response and any analogue or digital filters applied to the data).

4.3.1 decset.cfg

This file controls the windowing and decimation options. **decset.cfg** is the default name, but different file names can be specified in the **paths.cfg** file (see below). The standard file that we have been using is reproduced with explanations here:

	4	0				no. of decimation levels,	decimation level offset
	128	32.	1	0	0	7	4 51 1
1.0000							
	128	32.	4	0	0	7	4 51 4
.2154	.1911	.1307	.0705				
	128	32.	4	0	0	7	4 51 4
.2154	.1911	.1307	.0705				
	128	32.	4	0	0	7	4 51 4
.2154	.1911	.1307	.0705				

The first line gives the number of decimation levels and an offset to add to decimation level numbers - i.e. the first decimation level will be numbered $1+offset$. *offset* can be positive or negative, though in most cases *offset* = 0. But there are special cases where this might be changed. (As an example: Data sampled at 40 Hz at a permanent station is used for a remote reference for 10 Hz data for an MT survey. One way to do this would be to decimate the 40 Hz data by 4, and set *offset* = -1. Then decimation level 1 would be sampled at 10 Hz for both local and remote). For each decimation level there are two lines of parameters. For the first of these (e.g. the second line in this file, which begins 128 32. 1 ...) the parameters are, in order

- (1) Number of points in window (128 for all decimation levels here)
- (2) Number of points of overlap for adjacent sets (32.) (note that this is of type real and need not be a whole number; this can be used to keep data windows for different sampling rates "lined up". One of those features most users won't care to even think about.)
- (3) Decimation factor (1 for level 1 (undecimated); 4 for other levels)
- (4) Offset for centering decimation filter (0)
- (5) Offset for starting sets (0)
- (6,7) Parameters which define how missing data is treated (7,4)

- (8) Number of Fourier coefficients (FCs) to save; only the lowest frequency FCs will be output. In the example given here there will be 64 FCs produced by FFTing a single 128 point data segment. By setting this parameter to a value less than 64 , FCs which are worthless or redundant (due to overlap of frequency range covered by adjacent decimation levels) can be eliminated from output file. Also, in the example cited above with 40 Hz and 10 Hz data, one could set the number of FCs to save for the first decimation level (numbered zero after adding *offset* = -1; In this case the FT will be skipped for this decimation level). to zero.)
- (9) Number of filter coefficients for filtering before decimating to THIS level

The second line for each decimation level (3rd, 5th, 7th 9th lines in file) gives the filter coefficients for the digital low pass filters needed for filtering before decimation to THIS level.

The parameters are described more fully in the documentation for **decset.f**. To change the number of points in a set or the decimation factor between sets changes must be made to **decset.cfg**. Again, this file might seem pretty complicated, but most users will never have any cause to change very many (if any) of the parameters set in this file.

4.3.2 pwset.cfg

This file (but see below about using a different file name) is used to control pre-whitening options. A pi-prolate window is used for tapering the time series data before FFTing. This window does not always provide sufficient protection against spectral leakage without pre-whitening. The program allows three options for pre-whitening - (1) no pre-whitening; (2) first difference pre-whitening; (This works fine for long period data (e.g. 5s sampling rate, 20 - 10000s periods)); and (3) adaptive, autoregressive pre-whitening. Normally it is safest to just use option 3, with the length of the AR filter set to 3 or 4. The form of the file suggests that different options may be specified for different channels and/or decimation levels, but this has never been tested, is not currently supported, and does not seem worth fiddling with (but pre-whitening is definitely worth doing!). Note that the effects of pre-whitening are corrected before Fourier coefficients are output. Here is an example of this file.

```

4 5          number of decimation levels, number of channels
-1 -1 -1 -1 -1 One line for each decimation level, one number on each line for
-1 -1 -1 -1 -1 each channel.  each number gives instructions for pre-whitening
-1 -1 -1 -1 -1 the corresponding channel/decimation level.  If positive, this is
-1 -1 -1 -1 -1 the number of terms in the adaptive AR pre-whitening filter.
                  If 0 or 1, no pre-whitening.  If negative, first difference
                  pre-whitening is done.
```

NOTE: for wide band MT data I would not use first difference pre-whitening; instead I would use something like an AR 3 or 4 (with 128 point sets).

4.3.3 The system parameter file: test1.sp

This file is used to specify system parameters, including sampling rates, clock drift parameters, sensor orientations, electrode line lengths, instrument specific analogue filter corrections, and factors for conversion of data from counts to physical units. While the first two configuration files will probably be set up and left more or less the same for analyzing data of a particular type, the system parameter file will be specific to a particular installation of a particular instrument.

Note: The reformatting program for the EMI MT-1 system makes the .sp file automatically from the information in the data file header. The special version of **dnff** developed for the MT-24 system does not require a separate system parameter file—all information is read automatically from data file headers and run files.

Here is an annotated example of a system parameter file for an old EMSLAB MT installation at Valsetz, Oregon:

```
val031x          station/run ID
44.80  123.65    station coordinates: lat/long with decimal deg. frac.
19.5           geomagnetic declination of site
5             number of channels
16.0          sampling rate in seconds
0. 0.         clock offset & linear drift coefficients (cda, cdb)
H             channel id: H, D, Z for mag; E for electric
  0. 0.       sensor orientation; deg. E. of geomag N; vert. tilt
0.2441 2       count conversion (nT/count), number of filters
L2            filter type for first filter (L2 = 2-pole lo pass)
0.999 32.54 1.4254 filter parameters : gain, T0, alpha
L1            filter type for second filter (L1 = 1-pole lo pass)
1.00 .1905     filter parameters : gain, T0
D             Second channel - as above
90. 0.        sensor orientation; deg. E. of geomag N; vert. tilt
0.2441 2
L2
1.000 32.56 1.4252
L1
0.999 .1905
Z             Third channel
0. 0.         orientation for vert. - tilt to N, tilt to E (all geomag.)
0.2441 2
L2
1.000 32.45 1.4276
L1
0.997 .1905
E             Fourth channel : This is an electric channel; note difference
0.25374 313.6 0. 100 electrode line length (km), angle, tilt, amp gain
2.441 3       NOW count conversion [ mV/count], number of filters
L2
```

```

0.999 32.80 1.4201
L2
1.000 1. 1.414
H1          high pass, 1 pole
1. 174000
E          Fifth channel : still E; ALL electric channels are denoted by E
0.26696 49.3 0. 1
2.441 3
L2
1.000 30.64 1.4287
L2
1.0 1. 1.414
H1
1. 174000

```

Here is another more recent example, from a 5 component EMI system hooked up to a Quanterra 24 bit data logger.

```

PKD
0.0      0.0      lat./long.in deg
0.0
5          # of channels
1          sampling rate
0. 0.      clock offset & linear drift
Hx          Hx component
270. 0.      sensor orien. & vert. tilt
.0000025 1    count conversion(nT/count), #
of filter
TB
'bf4-9420.rsp'
Hy          Hy component
0. 0.      sensor orien. & vert. tilt
.0000025 1    count conversion(nT/count), #
of filter
TB
'bf4-9421.rsp'
Hz          Hz component
0. 0.      sensor orien. & vert. tilt
.0000025 1    count conversion(nT/count), #
of filter
TB
'bf4-9422.rsp'
Ex          Ex component
.10 270. 0. 1.0  electrode line length(km), angle,tilt,amp gain
.0025 1    count conversion, # of filter
TE
'ef-9309x.rsp' 1 3
Ey          Ey component

```

```
.10 0. 0 1.0    electrode line length(km), angle,tilt,amp gain
.0025 1          count conversion, # of filter
TE
'ef-9309y.rsp' 1 3
```

Note that channel IDs can be 1 or 2 characters. These IDs are passed on to intermediate and final results files. Some newer matlab programs for plotting results require the use of two character identifiers of the form Hx, Hy, Hz, Ex, and Ey to choose modes to plot etc. It is thus best to use this convention if possible. The first line after the channel designation gives the sensor orientation in terms of two angles—degrees East of geomagnetic north and tilt down. For the electric channels the sensor orientation is supplemented by (a) electrode line length and (b) amplifier gain.

For each channel you can specify a number of filters to correct for. In many cases all filters and calibrations will be combined into a single table, so that only a single filter (i.e., the combined table) need be corrected for. For each filter there are two lines in the test1.sp file. The first is a character*2 string which identifies the "filter type", and the second line contains any parameters needed to specify the filter response. This second line is read by the program as a character string; after figuring out the filter type, the program reads the parameters required for the particular filter type out of the character string. The program now knows about 7 kinds of filters. It should be easy to add additional/different filter definitions to the code. Here is a brief overview of how this works. The two lines for a single filter are read by the subroutine **getsp**. The first contains a character*2 string which gives the "filter ID" (this tells the program what type of filter to correct for. The second line, which is read in as a character*80 string, contains any additional information/parameters needed to complete specification of the filter. The routine **getsp** compares the filter ID to a list of what it knows about, and assigns a number in array **iftype** to identify the filter type. To add new filters, add if blocks to check for the new filter IDs. If the program doesn't know about a filter type given in the system parameter file, it prints a warning message, and proceeds (without making any corrections, of course). Routine **fcorsu** makes a table of frequency domain correction factors using the information from the system parameter file (along with other corrections for digital filters internal to the program).

For each filter type the routine expects certain filter parameters which it parses from the character*80 string (i.e., the second line of input for each filter). These parameters can be numbers (e.g., for example a corner frequency/period, gain, etc, as in the first example file where filter types like H1 (one pole HI-pass) and L2 (two-pole low pass) are used), or they can be character strings (e.g., a file name which identifies a sensor calibration file, as in the second example where the filter types are TE and TB, corresponding to EMI electric and magnetic field sensor calibration files; here the "filter parameters" are just the names of sensors files containing calibration tables (and for the E field sensors, some additional numbers to tell which columns of the table to read out of). To add new filters, add if blocks/read statements to read in the needed parameters for the new filter types. Using the parameters, **fcorsu** calculates a table of frequency domain correction factors. (NOTE: the correction factors output by **fcorsu** in array **rnrmt** will *multiply* the computed FCs). This overall correction table is calculated by multiplying together cor-

rections for all analogue filters and system responses, together with corrections for digital anti-alias and pre-whitening filters applied inside the program. Responses of particular filters are calculated by **afcor** for filter types (1-6) in the current version of the code, or by interpolating from a table using subroutine **rsptbl**, for several kinds of table formats (e.g., filter types TB and TE, used with sensor files from the EMI MT-1 system). The filter response computed by these routines is the output of the filter. The inverse of this response is incorporated into the combined internal **dnff** response table *rnrmt*. To add new filters, add code to analytically compute the response as a function of frequency in routine **afcor.f**, or to read a new kind of table using **rsptbl.f**, and modify **fcorsu.f** to incorporate the new kind of filter into *rnrmt*. NOTE: The Fourier transform done in **dnff** corresponds to an assumed time dependence of $e^{i\omega t}$. Make sure you use the same convention in defining real and imaginary parts or phases for any filter corrections.

The program currently knows about 7 general use filters or calibration files, and several specialized calibration files. General use filters include 1 and 2 pole lo and hi pass filters, a square "box car" type, and simple system response tables provided as a simple ASCII file. These are denoted respectively by L1 (one pole low pass), L2 (two pole low pass), H1, H2, (one and two pole high pass), BC (box car), AP (table has amplitudes and phases), and RI (table with real and imaginary parts). Depending on filter type, there are 1-3 parameters to specify on the line beneath that containing the filter designation. These are (1) gain (for L1, L2, H1, H2, BC) (2) T0 time constant for filter (L1,L2,H1 and H2); (3) alpha (L2,H2). For the BC filter there are 3 parameters - gain, width of box car, and offset of box car center from nominal sampling time. For the default look up tables the only parameter is the name of the file containing the table. The table file should be placed in a directory called **sensors**, in the working directory where the program is run from (or make a link in this directory).

Tables in the default format consist of a series of ASCII lines each with three numeric entries, in free format. The first entry is frequency (in hz), then amplitude and phase (for AP) or real and imaginary parts (for RI). All initial lines which start with a non-numeric character (0 - 9) on the first non-white-space field are assumed to be header/comment lines. From the first line which has a numeric character a three column table is read until EOF or ERROR. **rsptbl** tests for an increasing sequence of frequencies and reorders if this fails.

By modifying the subroutines **getsp** (get system parameters), **fcorsu** (set up filter coefficients) and function **afcor** (compute filter transfer function) any reasonable form of filter correction could be incorporated into the program. Several specialized filters of this sort have been added, including options to read EMI MT-1 B-field coil and E-field pre-amp calibration files. These calibration/filter options are denoted as TB and TE respectively. Parameters for these options are names of EMI sensors files, which again are to be placed in the **sensors** directory. There are also options for several other specialized look-up tables for calibration of particular instrument types (see routine **afcor.f**, for a brief description of what is allowed for).

4.3.4 The bad data file: test1.bad

This file tells the programs about "bad data". This file is optional. If it is not found the program prints a warning and proceeds under the assumption that there are no "bad data" segments. Data segments which are obviously contaminated by significant instrument malfunction, or other obvious problems, can be noted in this file so that they will be omitted from subsequent processing. (In the past, there has been a plotting program, running under X-windows which allows you to look at and mark bad sections of data files interactively; this is not currently supported, due to lack of funding.) In its current implementation, bad segments can be flagged with an integer 1, 2, 3, 4. These are supposed to mean:

- (1) Magnetics bad.
- (2) Electrics bad.
- (3) Long period bad.
- (4) All bad.

At a deeper level (in the program, I mean), there are two degrees of "bad" which data may achieve. The worst data (e.g., "ALL BAD") is not worth FFTing or saving ("DO NOT PROCESS"). Other segments may be obviously contaminated only in some channels, or period ranges (e.g. in the electrics but not in the magnetics). This data should be FFTd, but flagged ("DO NOT STACK") to warn subsequent processing programs (in particular, this data is stored with a negative set number). The mapping between the flag 1-4, and the classification ("DO NOT STACK" or "DO NOT PROCESS" is given in routine **mkbr.f**, in file **bdrcsu.f**. Currently, data classified "DO NOT STACK" is not used except for the remote reference. Of course all of this can be changed; note however that you also have to worry about telling the TF programs in the next step how to react to negative set numbers. If you don't like all of this complication you can: (a) ignore the whole business and use all data (OK if things aren't too noisy) or (b) use this option simply, e.g., flag all sections you don't like with "4".

Note also that the program allows one to specify an offset to *add* to the sample numbers in the input data file. This allows for one last chance for correction of clock errors before windowing of the data. Errors in the clock zero setting revealed by examination of plots can be corrected here. Making these corrections is essential for multiple station work (including remote reference if errors are very large) but not for single station processing. An example bad record file is given here:

```
2                                number of bad segments
416937.    416956. 1    beginning, ending sample numbers, iflag
426557.    428999. 3
0          offset to add to sample number on output
      (this last line is optional; offset is assumed to be zero if absent)
```

4.4 Running dnff

To run **dnff** you can put all needed input and configuration files in a single directory and run the program from that directory; the output will also be put in the current working directory. Alternatively, in the directory you run the program from you can place a file called **paths.cfg** (it has to have this name) which contains directory paths for where to find data files, system parameter files, bad record files, decimation control files etc.

Here is an example of **paths.cfg** file appropriate for the (typical) directory structure for the test data in **test**:

DATA	data directory
SP	system parameter directory
DATA	bad record file directory
CF/decset.cfg	FULL PATH NAME for decset file
CF/pwset.cfg	FULL PATH NAME for pwset file
FC	directory to put FCs in

For this example data files will be in **DATA**, system parameter files in **SP**, and output Fourier coefficient files will be put in **FC**. Leaving a line blank make the working directory be used for the particular (e.g., data or system parameter) file path. Note that for the configuration files (i.e., **decset.cfg**, **pwset.cfg**), the full path name (not just a directory) is specified. Of course the same directory may be used for several file types (e.g., data and bad record files in one directory), and of course the output directory (here, **FC**) has to exist before you can write a file in it. By specifying "standard" for the SP directory, you can make the program use a standard system parameter file, which should be in the working directory and called **standard.sp**.

Once all configuration, data, and system parameter files are set up (following the example in **test**) just type **dnff** to run the program. You will be prompted for input file names (relative to the **DATA** directory). After the run completes the program asks if you want to continue. Answer "y" to FFT another data file, "n" to terminate. Typically we use a list of data file names in a file called **dnff.cfg**

```
test1.bin
y
test2.bin
n
```

to FFT a series of files in a "batch" mode with **dnff < dnff.cfg**.

4.4.1 Simple ASCII input files

By default the program assumes that input files are in the standard binary format. To read from simple ASCII files use the command line option **-a** or **-A**. In both cases the program expects a simple ASCII input file with *nch* channels of integer data on each line. Only very simple ASCII input files are supported at this time. The program reads with free-format, so all numbers on a line should be separated by blanks. (This could be easily

changed to a fixed format.) The program also requires that all data be consecutive (no gaps allowed), and that there are no sample numbers in the file (only the actual data).

To use the ASCII format, you also have to provide the same clock reset information required by **rfasc**. With the the **-a** option this information is provided by a clock file of the same form used by **rfasc**:

```
16.0          <=== sampling rate, in seconds
85 07 20 0 0 0 <=== clock reset: yr,mo,day,hr,min,sec
85 07 20 0 0 0 <=== universal clock zero: yr,mo,day,hr,min,sec
```

Here the clock file must have the name **test1.clk**—i.e., the name is made from the root of the input data file name specified by the user when prompted by **dnff** with the **.clk** suffix added. With the **-A** option the same three lines given above are read from the first three lines of the data file, and separate clock reset file is not required.

Note that for the test data in **test**, you should get exactly the same results from

- (1) Running **rfasc** on **test1.asc** to make binary file **test1.bin**, then running **dnff** on **test1.bin**; or
- (2) running **dnff -a** directly on **test1.asc**.

The modifications to **dnff** to allow direct reading of these simple ASCII files is a simple example of how this program can be adapted to directly read other file formats. This is discussed further below in section 4.6.

4.5 Format of output FC files

The output of **dnff** is a binary fixed record length direct access Fourier coefficient file ordered by frequency.

The structure of this file is:

Decimation

Level #1

```
|
|
|
|-----|
|               Header record for frequency #n
| Fourier      Record #1
| Coefficients      .
| for frequency      .
|   # 1              .
|               Record #Nn
|-----|
|-----|
|
|   A similar block is repeated for each decimation level
|
```

These Fourier coefficient files can be used by all subsequent processing programs. The Fourier coefficient files are called `test1.f#` where `#` is an integer giving the number of channels of data. Note that the Fourier coefficients are kept in the original measurement coordinate system, with all filter corrections, count conversion etc., applied. Magnetic field Fourier coefficients are output in units of $nT(hz)^{-1/2}$ and electric field FCs are output in units of $(mV/km)(hz)^{-1/2}$. The header contains all necessary information needed to correct for sensor orientations, local magnetic declination, etc (provided the user inputs this info!).

By default the FCs are stored in a compressed 2-byte floating point binary format. The range of the floating point representation is determined automatically (and separately for each frequency) by the output program. When read by the proper routines, the FCs are complex numbers with proper physical units. This compression of the data can be suppressed by setting the logical parameter `lpack` to `.false.` in the include file `iosize.inc`. With `lpack = .false.` FCs are output as standard 8 byte complex numbers (i.e., two 4 byte reals). See the section on making `dnff` and changing parameters above. NOTE: if you change `lpack` here, you need to change it also for programs that read the FC files (i.e., `tranmt` and `multmtrn`).

4.6 Changing input format for data files

For efficient routine processing it is desirable to eliminate the first (reformatting) step in the processing programs. With minor changes `dnff` can be adapted to read directly from almost any sort of input data files. The command line option for reading simple ASCII data files provides an example of how this can be accomplished. These modification will be easiest if you can guarantee (as we have assumed for the ASCII files), that:

- there is no missing data in the input files
- all channels are "multiplexed" in one file, and
- system parameter information will be provided in a separate file.

To allow for missing data, get system parameter information from headers, or allow for merging of channels from multiple files more significant changes to the code will be required. For examples of these sort of changes, you can get special code for mini-SEED or EMI MT-24 data files from egbert@oce.orst.edu.

The standard subroutines for data input are all in the file **inpu_bin.f**. There are 3 main routines in this file. To change input format, these routines will need to be modified, replaced, or supplemented by other routines.

- (1) **cininit** : This initializes IO by setting up path names for configuration files, system parameter files, data files, etc.
- (2) **rdhd** : This reads the binary file header to find out how many channels of data there are.
- (3) **rdblk** : This routine gets a chunk of data with consecutive sample numbers from the input file. A reasonable sized chunk would be approximately equal to the number of points in the first decimation level, but the exact number of points is unimportant, as long as sufficient storage is allowed for in all necessary arrays. [BUT NOTE: the program expects a series of integer samples which are sequential. Each sample returned by **rdblk** consists of $NCH + 1$ channels - the first channel is the actual sample number. If the data in the input file is not in this form, the reading routine will have to put it in this form before handing the data over to the main program (e.g. if there are no sample numbers in the file, the reading routine should add them).]

Note that the current version of **rdblk** for binary files deals with data gaps. In particular, the routine fills in small data gaps with a missing value code (but with the correct sample number, so that there are NO GAPS IN THE SAMPLE NUMBERS HANDED TO THE MAIN PROGRAM). If a very large gap in sample numbers occurs in the input data **rdblk** does not fill in but returns an error condition (see below). If you replace this routine with another and you expect that data gaps may exist in files you want to process, the replacement reading routine (and the way it interfaces with the program) will have to be designed to deal with this problem. If you don't expect any gaps a very simple reading routine would work - just read the next n data points, with sample numbers.

The reading routine is called in two places. The initial call occurs before execution of the "main loop", subsequent calls occur at the end of this loop. With the current setup there are a number of test statements after the later call of **rdblk**. These test the variable **ierr** which can be 0, -1, -2, or -3. If **ierr** = 0 the read was normal; if **ierr** = -1 the end-of-file has been reached and the program exits the main loop; if **ierr** = -2 the sample numbers are not in increasing order and the program terminates; if **ierr** = -3 a large gap exists

between the first sample of the current block and the last sample of the previous block. In this case the program essentially starts over on windowing and filtering the data; no data windows will contain the gap.

Again, if there are no data gaps a simple reading routine could replace **rdblk**. To allow for gaps in the data something which emulates some of the complexities of **rdblk** will be necessary.

To read the simple ASCII files the following changes to the routines in **inpu_bin.f** and **dnff** were made.

- (1) New logical variables were added to **input.inc** to keep track of the ASCII options requested.
- (2) Additional code was added to **cininit** (in **inpu_bin.f**) to initialize reading from ASCII files (i.e., if *l_asc* = *.true.* different instructions for opening and reading headers of files are executed).
- (3) A new subroutine **rfasc** was added (in **asc_rec.f**) to read clock info from the **test1.clk** file (for the **-a** option) or from the input data file header (for the **-A** option), and initializes starting record numbers, and the number of data channels *nch*. This routine takes the place of **rdhd** for binary files.
- (4) A new basic reading routine **rdasc** was added (in **asc_rec.f**). This is used in place of **rdblk** with both **-a** and **-A** options, and has essentially the same function as **rdblk**.
- (5) The main program (**dnff**) was modified to parse the command line for **-a** or **-A** options and to call **rdasc** instead of **rdblk** when these options are requested.

Similar modifications of the source code would allow other simple formats to be accommodated.

5 Transfer Function Estimation Program

The transfer function program (**tranmt**) computes robust single station and remote reference transfer functions between a pair of local reference channels (generally two orthogonal horizontal magnetic field components) and some number of other channels. The general philosophy behind the estimation scheme is described in Egbert and Booker, 1986. In addition there is an automatic "leverage control" feature (e.g., Chave and Thomson, 1989), and an option to use a hybrid coherence sorting/regression M-estimate, as described by Egbert and Livelybrooks. Error bars are computed using the asymptotic approach described in Egbert and Booker (1986).

5.1 Input Files

The input expected by this program is one or more of the FC files output by **dnff**. It is possible to have FCs for local and remote sites in the same or different FC files. Similarly, it is possible to have electric and magnetic channels in the same or different FC files. Also **tranmt** can merge FC files from a series of sequential runs into a single TF estimate (i.e., if there are three separate runs of a particular sampling band, these can be run separately through **dnff**, and the three resulting FC files can then all be used by **tranmt**).

In addition to FC files there are 3 required configuration files, plus one optional file. The three required files are generically called **tranmt.cfg**, **options.cfg**, and **bs.cfg**, but any names (or suffixes) can be used. Roughly, **tranmt.cfg** tells the program which data files to use, (and which "options" file to use), the options file **options.cfg** specifies processing options, and the band setup file **bs.cfg** tells the program which frequency bands to compute TF estimates for. There are simple default versions of these required files in **test**. The optional configuration file **ref.cfg** is used to change default definitions of reference and predicted channels (see below).

5.2 Making tranmt

There is a simple UNIX makefile; just type "make tranmt" to make the executable, then "make install" to move the executable to BIN_DIR (set at the factory for testing to **test/bin**). There is one include file which you might want to edit: **tranmt.inc**. This file is in the tranmt source directory. Following is a brief synopsis of parameters which might need to be changed for some purposes. Any parameters in the include files not explicitly discussed here should be left set as they are.

nstamx

Maximum number of "stations" (i.e., channel groups). Note that a here station refers to the set of all channels grouped together in a FC file. There could be several FC files (same group of channels, but a different run) for one station. There could also be several channel groups used to estimate a local impedance tensor, e.g., if **E** and **H** were in separate FC files.

nchmx

Maximum number of channels for a single channel group.

ntfmax

Maximum number of data points to allow for for a single frequency band. Increase for very long time series. It also might be necessary to decrease the size of this to get the program to fit into memory, particularly if parameters like *nstamx* and *nchmx* are large.

ncbmx

Maximum number of data points to allow for in wide coherence sort band. If coherence sorting is not used this can be equal to *ntfmax*. If coherence sorting is used this might need to be roughly 3 times as large as *ntfmax*.

nsetmx

Maximum number of sets (i.e., time windows totaled over all decimation levels) to allow for. This should be roughly one third of *ntfmax* (at least for the way I use the program typically).

nbmax

Maximum number of frequency bands to compute estimates for. Increase this if you significantly increase the number of frequency bands in the **bs.cfg** file.

ndmax

Maximum number of decimation levels to allow for. Should be set to whatever is used for **dnff** (compare to *ndmx* in **D/params2.inc**).

nfreqmax

Maximum number of FCs saved for a single decimation level. For example, with 128 point sets this would be at most 64, but is often set to less. Again, this should be set big enough to be consistent with the number of FCs being save by **dnff**.

ntpmx

Maximum number of FC files for a single station. Each FC file would correspond to a different "run" of the same set of data channels. Often this could be set to 1.

lpack

This is a logical parameter set to *.true.* when the FC files are stored in packed integer format. In this format one complex number is stored in 4 bytes. Set this the same way it is set for **dnff** in **D/params1.inc**. Packed format is the normal usage. However, for data recorded with 24 bits resolution, you *might* want to use standard binary format (i.e., set *lpack* = *.false.* in both **dnff** and here.) However, I do not think this is necessary in most circumstances.

lfop

This is a logical parameter normally set to *.false.* Setting this to *.true.* makes the program open and close all FC files before and after every read. This is necessary on some systems when the number of stations in the array is too large, since some systems limit the total number of files that may be opened in a Fortran program. (This should never need to be changed for **tranmt**; but it might need to be changed for **multmtrn**.)

5.3 Testing tranmt

You can test **tranmt**, after successfully making one or more FC files. Here we assume that two FC files have been made: **test1.f5** and **test2.f5**, and that both files are in **test/FC**. Look in the main **tranmt** configuration file **tranmt.cfg**. The file is set up to do three runs: First single station processing of the first "site" TS1 (i.e., the synthetic data in **test1.asc**); single site processing of the second site (data from **test2.asc**); and remote reference processing using site two as a reference for site one. After completion of the run there will be three files in **test/MT**: **test1.zss**; **test2.zss**; and **test1r2.zrr**. These final results should agree with the corresponding results in **results**. To plot results in the files you can use the matlab program **apresplt** described in document *Matlab M-files for EMTF and multmtrn* in **doc/PS/matlab_doc.ps**.

5.4 Configuration files

5.4.1 Main configuration file tranmt.cfg

This file tells the program which Fourier coefficient files to process. The program prompts for the file name, which is arbitrary.

An example control file called **tranmt.cfg** is given here:

```
test          <----- station/run name : used for making output file names
options.cfg   <----- the options file for this run
1             <----- number of groupings used for FC files
1 5           <----- for group 1 : # of FC files (runs), # of channels in group
test.f5       <----- name of first FC file [ this line could be repeated if the
                  number of FC files is greater than 1 ]
n             <----- NO, do not continue with another set; if the entry is 'y',
                  all of the above lines should be repeated with appropriate
                  file names etc.
```

Some further notes on this control file, keyed to the line numbers: (Refer also to the more complicated example below)

- (2) The *options file* controls various processing options. This is the second required file, discussed below.
- (3) Channel groupings refer to the way channels are grouped together in FC files. In most cases channel groupings correspond to stations, so the number of groupings will be 1 for single station data. However if **H** and **E** are in separate files, the number of groupings would be 2. If local **H** and **E** were in one file but the channels to be used for the remote reference were in a separate file, the number of groupings would again be 2. If **H** and **E** and the remote were in separate files this number would be 3. In principal, each channel could be in a separate file. Note that the maximum

number of channel groupings allowed by the program is referred to in the code as *nstamx*, and that channel groupings in the code are referred to as *stations* (consistent with the initial, and still most common, usage.) Note also that the number of files does not in any way determine if the processing is remote reference. This is set in the **options.cfg** control file described below. By default the program orders the channels sequentially, following the order of channels and beginning with the first file, then adding the channels for the second channel group, etc. The first two channels are then assumed to be the local reference. If remote reference estimates have been requested, the first two channels from the last station (channel grouping) are assumed to be the reference channels. These are the default conventions, which can be changed by using the command line options **-s**, as discussed below.

- (4) For each channel grouping there are 2 (or more) lines. This is the first line for the first (and in this case only) grouping and tells: (a) The number of FC files for the first channel group. Here there is only 1, but it is possible to have multiple Fourier coefficient files corresponding to the same set of components (i.e., multiple runs of the same site/array could be in separate FC files). Note that the grouping of channels must be the same for all runs. (b) The number of channels in this group.
- (5) The file names for the first channel group are now given, 1 per line. This line is repeated for each FC file for this group. Note that the file path is given relative to the FC directory specified in the options file. Lines (4-5) are repeated for each group, if more than one grouping is used.
- (6) "y" or "n" : process another station or not? If "y" is specified, continue with another set of instructions (lines 1-6) for another run (e.g., different station or sampling band, or different processing options).

Another example, this time more complicated. Now remote reference transfer functions are to be estimated. (So the options file described below should be changed to reflect this!) The Fourier coefficients for the local station (VAL in this example) are in three files. Those for the reference channels are in two files. There are overlaps between all files from these very long period MT sites. As long as the sample numbers in the original time series were correct (i.e. were relative to the same universal clock zero for all files) the program will match up FCs for all of the appropriate data segments. Note that the reference station here is just another 5 component station. Only the first two components for the reference station will be used in the remote reference transfer function processing.

```
tstrr          <----- station name;
options_rr.cfg <----- the options file for this run (with RR turned on)
2             <----- number of channel groups
3 5           <---- for group (station) 1 : number of FC files (3), number of channels
val3a.f5      <---- name of first FC file
val3b.f5      (3 file names, one to a line)
val3c.f5
2 5           <---- for group (station) 2 : number of FC files (2), number of channels
mon12a.f5     <---- 2 file names - 1 to a line
```


mon12b.f5

```
n      <----- NO, do not continue with another set; if the entry is 'y',
           all of the above lines should be repeated with appropriate
           file names etc.
```

5.4.2 Options File

This is a file which tells the program which processing options to use. The idea is that one might want to process a series of stations/runs with the options set the same. For example, routine robust remote reference processing of two 5 channel MT sites could always use the same options file. The pathname of this file is specified in the main control file (see above), so the name of this file is arbitrary.

Here is an example options file

```
Robust Single station  <--- this is a header which is added to output files
F5TEST                 <--- Input (FC) directory
MT                     <--- Output directory; put impedance etc. files here
CFTEST/bs_nod          <----- full path name of band set up file
y                       <--- 'y' for robust, 'n' for LS
n                       <--- 'y' for remote reference, 'n' for single station
n                       <--- 'y' for e field ref, 'n' for magnetic
n      <-----      output coherence vs set no. (if yes provide file name on next line)
0. 0. 0. 0. 0         coherence sorting parameters **** see below*****
```

Here are some further notes on the options file, keyed to line numbers.

- (1) The first line just gives a character string which is written into output files as an identifying header
- (2-3) The next two lines give relative pathnames for input and output files.
- (4) Line 4 gives the the pathname to the band setup file discussed below.
- (5-7) Lines 5-6 set processing options with 'y' for yes, 'n' for no.
- (8) This is more or less a debugging/testing feature. Probably best to leave this set to 'n' . If output is requested, the next line should give a file name. See code for details about this output file.
- (9) Coherence Sorting Parameters: Four real numbers and one integer are required in general. These are (in order):

coh_target, coh(1), coh(2), coh_min, nu_min

These parameters are used to specify coherence cut off levels for coherence pre-sorting. In this scheme coherence is calculated for wide frequency bands for each

time segment, and only time segments which achieve a specified minimum coherence are used for further processing. This feature is most useful for single station dead-band data where noise in the magnetic components can be large enough to cause serious bias problems with single station estimates. See Egbert and Livelybrooks, Geophysics, 1996 for further discussion and justification. The five parameters specify a scheme for determining coherence cut off levels, which adapts to the number of data points and typical coherence levels. The scheme tries to trade off (in an ad hoc manner) between reducing bias and variance - i.e., we want to use only segments of "high enough" coherence, but at the same time keep a reasonable number of degrees of freedom in the estimates. If all parameters are zero, this feature is inoperative. If you use this feature, you will have to experiment with different parameter values to get a reasonable tradeoff between keeping enough data, and getting rid of enough noise.

Here is the meaning of the 5 coherence sorting parameters :

- (1) *coh_target* is the target coherence; ideally we would like all time segments to achieve this coherence. (e.g., *coht* = .95)
- (2-3) *coh_p* two real parameters used to determine a target number of degrees of freedom in the transfer function estimate via :

$$nu\_target = coh\_p(1) * nu^{**} coh\_p(2)$$

where *nu* is the number of points available (e.g.: *coh_p*(1) = 3., *coh_p*(2) = .5)

- (4) *coh_min* minimum acceptable coherence level. (e.g., .8)
- (5) *nu_min* - minimum number of data points (e.g., 20)

If possible we would like to use at least *nu_target* data points, all from time segments with coherence at least *coh_target*. If there are not *nu_target* points in sets with coherence above the target coherence, we accept lower coherence sets, until *nu_target* points are available, or until *coh_min* (the minimum acceptable coherence) is reached. In general we don't accept sets with lower coherence unless this is necessary to get the minimum number of data points, *nu_min*.

NOTE : E-field reference hasn't been used in a long time, and might not work now .... I'll try to check out and fix these things soon. .

5.4.3 Band Set Up File

This tells the program which frequency bands to produce estimates for. This file is required. The name of the file is given in the options file, so the form of the file name is arbitrary.

An example band set up file:

	No. of bands
28	
1 34 45	band 1: dec. level; band limits (lo & hi); weight parameter
1 27 33	
1 21 26	
1 16 20	
1 13 15	
1 10 12	
1 8 9	
1 6 7	
2 18 22	
2 14 17	
2 11 13	
2 8 10	
2 6 7	
2 5 5	
2 4 4	
3 11 13	
3 8 10	
3 6 7	
3 5 5	
3 4 4	
4 11 13	
4 8 10	
4 6 7	
4 5 5	
4 4 4	
4 3 3	
4 2 2	
4 1 1	

Note that this file has the frequencies ordered from highest to lowest with no overlaps (i.e. a reasonable decimation level has been chosen for each frequency band). In the output file, results for the bands will be printed in this order. By varying the form of this file any bands can be printed out in any possible order.

NOTE: The form of this file has changed slightly recently. There used to be an extra entry in each line, which was used to define data weights which depended on power. This feature is not of much use for MT data, and has thus been eliminated. Old band set up files with the extra entry for power dependent weights (usually these were set to 0) can still be used, but the weight parameters are now ignored.

5.4.4 Reference File

This optional control file can be used to change the default rules for deciding which channels are predictors, predicted, and remote reference. **tranmt** makes a list of all

channels, beginning with the first station grouping in the **tranmt.cfg** file, keeping the order implicit in this file. Additional channel groupings are added in order after this. By default the local and remote reference channels are chosen from this list as follows: local reference channels (i.e., two predicting channels for the transfer functions) are the first two channels in the list; remote reference channels (if appropriate) are the first two channels in the last group. These defaults make the most common usage of the programs easy: Channels are grouped by stations and the first station is the local site, while the second is the remote. *This convention requires that the local magnetic channels be the first two channels in the time series files input to **dnff**.* By using the **-sref.cfg** command line option these defaults can be changed, allowing different channels to be used for local and remote references. The file name **ref.cfg** given as the argument to the **-s** option defines the reference channels.

Here is an example of this file:

```

3 4          <====    local reference channel numbers
2            <====    number of output (predicted) channels
6 5          <====    predicted channel numbers
7 8          <====    remote reference channel numbers

```

In this example, channels 3 and 4 (order in the list of all channels) are to be used for the predicting (local reference channels). The overall channel order which we refer to here is determined from (a) order of channel groupings as listed in **tranmt.cfg** file, and (b) the order of channels within each FC file. Transfer functions for two predicted channels (6 and 5) are requested, and channels 7 and 8 will be used for the remote. Note that if remote reference estimates are not requested, the last line of this file is not needed.

5.5 Output

Currently **tranmt** outputs two files: A file containing cross-spectra for all predicted and reference channels, and a "Z-file" containing local TFs and error bars. Separate files for MT and GDS parameters in one or more fixed coordinate systems (as output by previous versions of **EMTF**) are no longer produced. The Z-file for our example here would be called **test1.zss**. Z-files contain TFs (either single station or remote reference) between a pair of local channels and 1 or more predicted channels (usually H_z , and/or E_x and E_y , but the file format supports more general array configurations). All TFs are in the measurement coordinate system. The Z-files also contain channel information (orientations, channel names, etc.) and the full covariance of signal and residuals. With these matrices it is possible to correctly compute error bars in any coordinate system. The multiple station program **multmtrn** also outputs Z-files of the same format. The Z-files produced for single station, remote reference, and multiple station processing are interchangeable. The same calculations are applied to the contents of any of these files to change coordinates or compute error bars, so this format makes it simple to combine TFs from single site and remote reference processing. The format of the Z-files, and instructions for rotating and computing error bars are given in the separate document "**Errors Bars for Transfer Function Elements in Z-files**".

5.6 Likely Problems

There are three things likely to cause problems: (1) I/O (We have had to make some minor changes almost every time this code is ported to a different machine. These changes should be pretty minor. For example an IBM running AIX won't allow "end=" in IO statements, but instead requires "err=" or "iostat=", but other systems might demand "end=", etc. (2) A different level of tolerance for minor syntax errors in your system. Some systems are very picky, others less so. (3) Parameters which set up array sizes may not be big enough for some applications. We have tried to put in some checks, but so far not everything is checked. Look at the first few lines of the main programs, and various include files (e.g., `datz.inc`) to see if parameters are set properly for the data set you are trying to process.

6 Plotting of Apparent Resistivities and Phases

Some scripts are provided for reading in the Z-files, converting to apparent resistivity and phase, and plotting in `matlab`. A more complete description of some `matlab` tools which have been developed for working with output of these `multmtrn` programs are provided in the separate document "Matlab M-files for EMTF and `multmtrn`", in `doc/PS/matlab_doc.ps`.

7 References

- Chave, A.D., and D.J. Thomson, Some comments on magnetotelluric response function estimation, *J. Geophys. Res.*, **94**, 1989.
- Egbert, G., and J.R. Booker, Robust estimation of geomagnetic transfer functions, *Geophys. J. R. Astron. Soc.*, **87**, 173-194, 1986.
- Egbert, G.D. and D.W. Livelybrooks, Single station magnetotelluric impedance estimation: coherence weighting and the regression M-estimate, *Geophysics*, **61**, 964-970, 1996.
- Egbert, G. D., Robust multiple station magnetotelluric data processing, *Geophys. J. Int.*, **130**, 475-496, 1997.

Appendix C: MATLAB Plotting Programs for MULTMTRN and EMTF

Matlab M-files for EMTF and multmtrn

Gary D. Egbert*

February 27, 1998

1 Overview

This document describes some of the more commonly useful matlab scripts and functions (M-files) which have been developed at OSU for post-processing and plotting of results output by **EMTF** and **multmtrn**. In general we have found that there are too many variants on instrument and survey configurations (and on interpretation philosophies) to compute all reasonable interpretation parameters (e.g., apparent resistivities and phases in specific coordinate systems, parameters like skew, etc.) in the transfer function programs. What makes sense to compute, and how to compute it, depends too much on details of how the data were collected, and what the user intends to do with the results. The same general considerations apply to outputs from the multiple station program (only here the multiplicity of possible array configurations is much greater, and the variety of parameters and diagnostics that might be plotted or otherwise looked at, becomes very unmanageable). We have thus chosen to stick to relatively simple outputs from the transfer function programs, and then further process and plot these results in **matlab**. For example, the principal outputs of the single station and remote reference processing program **trnamt** are the Z-files which contain transfer functions and error covariance matrices in the measurement coordinate system. Rotation into other coordinate systems, conversion to apparent resistivities and phases, merging of different sampling bands and site into pseudosections, reformatting results for input to other inversion programs, and ultimately other post processing such as distortion analysis are then done in **matlab**. Similarly, output from the multiple station program **multmtrn** has been reduced to a few basic files (a spectral density matrix file, a series of Z-files giving local TFs, and a plane wave array TF file). These files can be processed further in **matlab** to make a variety of plots, and to explore the multivariate structure of the signal and noise.

Scripts and functions in **matlab** can be easily tailored to accommodate specific survey configurations and individual preferences. To facilitate this we provide a series of matlab functions which do some basic operations like reading TF output files, rotating coordinate systems, and some basic plotting functions. The most important of these are documented below. We also provide a limited number of higher level plotting programs which do some of the most common things, such as plotting apparent resistivities and phases, and

---

*College of Oceanic and Atmospheric Sciences, Oregon State University, Corvallis; egbert@oce.orst.edu

plotting array results from the multiple station program. These programs provide simple graphical interfaces for common plotting tasks. However, some aspects of these programs may need to be changed for particular applications. Generality *and* ease of use are hard to come by, and expensive to realize in software!

We begin with a summary of the usage of a few common higher level plotting programs, and then summarize some of the most important component scripts, which might be useful for constructing other specialized applications. The higher level **matlab** functions and scripts provide examples of the use of these lower level more general functions.

To use the matlab plotting and post-processing functions you need to put all of the directories you need to use into the **matlab** search path. This can be done on a unix machine by adding the names of these directories to the definition of the environment variable **MATLABPATH**. You can also use the *path* command in **matlab**. Note that the higher level routines call functions from other sub-directories like **matlab/IN** and **matlab/UTIL**, so it is easiest to just add all subdirectories to the path.

Most of the scripts (in particular those needed for standard apparent resistivity and phase plotting) will run under **matlab** versions 4 and 5. A few (in particular **Pw_plot** will only run under **matlab 5**. The scripts have been tested to run on both a SOLARIS 2 (Sun Unix) platform, and a PC running Windows95. There are a few features which act funny in Windows95. More generally, plots and dialogue boxes appear differently on different computers. Furthermore the matlab plotting scripts open windows at specified locations on the screen (of specified size), and the appropriateness of these choices depends on the size of the monitor (and the platform). To get around this problem in a simple way, all sizes and locations of plotting windows are set in file **UTIL/stmonitr.m**. In my case, where I switched between a portable PC with a very small screen, and a UNIX workstation with a 17 inch monitor, I set the appropriate parameters by testing for the computer platform (using the matlab function **computer**). Others will have to change this file to fit their particular situation.

2 Higher Level Plotting Routines

There are several higher level plotting routines, which allow the user to interactively choose files to plot, and then modify/extend the plots in various ways. These routines call input routines from **matlab/IN**, and also some computation routines.

2.1 apresplt.m and ZPLT

The highest level program for plotting apparent resistivities and phases is called **apresplt**. This script, and most of the specific functions called by this are in subdirectory **ZPLT**. In the simplest application of this plotting routine, just type **apresplt** while in matlab. A dialogue box will appear for specifying the Z-file to plot results from. Pick the desired file. **z_mtem** is now capable of plotting results from either a conventional MT sounding (with two orthogonal electric field dipoles) or from a profiling setup with multiple dipoles. In the latter case, all ρ_a and ϕ curves corresponding to E_x are plotted in one figure (with different colors for each E_x dipole), and all E_y curves are plotted in a separate adjacent

figure. In the case of a conventional MT setup the two principal modes are plotted on the same figure. The program makes the choice of which to do automatically. (But it's difficult to guarantee that this will work in every case!)

2.1.1 Plot Options

The apparent resistivity plotting program has several options which can be activated by clicking on the "Plot Options" button on the upper menu bar of the left-hand figure. These allow you to :

- (1) Modify plotting limits, which are normally chosen automatically. (Note: the program tries to keep a 1:1 aspect ratio on a log scale for the apparent resistivity plots.
- (2) Rotate into a different coordinate system. Initially, results are displayed in measurement coordinates. Note that for profile data there is some ambiguity in how rotation should be done, since rotation requires that channels be grouped into pairs. The program follows a set of rules to try and pair off E_x with E_y . (We assume that the order of channels in the FC files reasonably reflects spatial proximity ... but this may not always be correct). If there are extra components of, say, E_x (as there typically are) the program uses some E_y components for more than one pair. For example if a profiling setup consists of 5 E_x and 3 E_y components, the program will make 5 pairs, and, after rotation a total of 10 curves will be plotted (5 in each figure). If there are no E_y components, obviously rotating the data makes no sense, and will result in some kind of error. For conventional MT rotation is of course straightforward.
- (3) Results from additional sampling bands can be added to the plot, by clicking the "add band" button. One problem with doing this in general, is that there will often be significant overlap between bands, with one band having very poor estimates (usually this is the higher frequency sampling band for which comparatively little data is available). Thus ideally, one would like to splice together the different sampling bands without (much) overlap. To do this in a completely general way is difficult. By default we have things set up to just plot all of the results from all bands, and we provide a somewhat general approach for defining fixed choices of frequencies to plot for each sampling band. We explicitly provide the code for a specific example set of bands, which was used for a wide band EM profiling survey in which data was sampled in three bands. The code provided is very specific to the particular sampling bands, processing options, and output file naming conventions used for this particular survey. By modifying the simple driver routine **apresplt.m** and/or one other file (**mkpltind.m**) something similar to what we have done for this specific example could be easily developed for different circumstances. Note that to use the scheme we describe here you have to uncomment a couple of lines in **apresplt.m** (see the source file in **ZPLT**); by default **apresplt** comes set up to plot all data from all bands.

Here is the example. There are three bands: a mid-band ($M = 960$ hz), a low band ($L = 120$ hz), and a very low band ($V = 3.125$ hz). The band for each Z-file was identified by the seventh character in the output Z-file name; thus it is possible to identify which band a particular file corresponds to from the file name alone. The idea is that we make a function (M-file) called **mkpltind**(*file_name*) which figures out from the file name which band the file corresponds to, and then returns a set of 0/1 indicators for each frequency in that band (0 means don't plot that frequency, 1 means plot it). In our example the choice of bands to plot is hard coded in this function. **mkpltind** is then used in **apresplt.m** to define the character string **MKPLTIND** = ['pltind = [pltind ; mkpltind(cfile)];']; Then, in the main plotting routine **z_mtem** :

```
eval(MKPLTIND)
```

is executed for each file to concatenate the indices of frequencies which should be displayed into a list *pltind* for the sequence of all files loaded. All input arrays (TFs and error covariances) are also concatenated into arrays which include all frequency bands from all sampling bands. Then, *pltind* is used to determine which frequencies are plotted.

By making a function like **mkpltind** which can figure out which band has just been read in and then which frequencies to plot, and substituting this in the definition of **MKPLTIND**, you can change the way frequencies are chosen for splicing bands together. Note that it is not necessary to name this function **mkpltind**. Thus you could keep several versions of these "plotting index" functions around, and just change the definition of **MKPLTIND** in **apresplt** to switch between versions. The simplest way to merge sampling bands in the plot is to just plot all bands (the way the source code comes in the tar file). In this case the making of array *pltind* is very simple, and does not require a special function like **mkpltind**. Just define

```
MKPLTIND = ['pltind = [ pltind ; ones(nbt,1)];'];
```

to make a string of ones of length equal to total number of frequency bands in all sampling bands.

2.2 sdm_plot.m

This script and the specialized M-files that it calls are in subdirectory **SDM**. The command to execute **sdm_plot** is a matlab script (not a function) which plots various diagnostics which can be computed from the spectral density matrix (SDM) output by the multiple station program in the S0-files. The script is fairly elaborate, and it almost certainly has lots of bugs which will show up as it is tried on different platforms/data sets. So far the program has been tested primarily on small 5 component MT arrays (2-3 station). There might be problems with some features if you try to use this with very different types of arrays. At the very least, different sorts of arrays (e.g., an array of E-field dipoles) would benefit from some different defaults and plot types.

When you type **sdm_plot** the script opens a dialogue box for browsing for SDM files. After you choose an S0-file, a plot of normalized SDM eigenvalues vs. period will appear. There are two specialized menus at the top of the plotting window called **plot options** and **Eigenvectors**. These menus contain buttons which generate several types of plots from the SDM. NOTE: In addition to the sdm file named **array_name.S0** this plotting program reads the ASCII file named **array_name.SN**. The program derives the name of the SN-file from the name of the chose S0-file. Both must be in the same directory (as they would normally be after running **multmtrn**).

There is also a slider bar beneath the period axis on the eigenvalue plot. With the slider, and the "PLOT" pushbutton you can plot the eigenvectors. The **Eigenvectors** option in the top menu, opens a dialogue box which allows you to modify the appearance of the eigenvector plots, including choosing the number of vectors to display, scaling options, and the way components of complex vectors are displayed (as real/imaginary vectors, or polarization ellipses). These plots consist of magnetic and electric vectors plotted on a map of station locations. Currently H_z components are not displayed. If station coordinates are not in the SDM files (they won't be if they are not provided to **dnff** when the FC files are made), the local vectors are plotted on a diagonal line at the points $(1, 1), (2, 2), \dots (nsta, nsta)$. Also note that this sort of plot will only make sense for plotting pairs of horizontal components. Currently the program pairs off horizontal channels of the same type (i.e., H_x with H_y and E_x with E_y). Only channels from the same station (more precisely: FC channel grouping) can be paired off. If there are multiple channels of one type in a single channel grouping (as there would be for EM profiling data), the program just takes the first of the type that it finds. This is done by **ch_pair.m**. More specialized plotting routines may be useful for plotting more general array configurations, such as profiling data. For now, plotting of eigenvectors is limited to more conventional MT arrays.

To plot electric and magnetic vectors on the same plot, some sort of scaling into common units is required. This scaling is defined by a reference resistivity, ρ_{ref} . If the actual apparent resistivity has equals ρ_{ref} , then H and E vectors (or polarization ellipses) will have the same length on the eigenvector plots. This reference resistivity can be changed in the menu. There is also an option to display everything in non-dimensional SNR units—i.e., with each channel divided by the estimated incoherent noise powers for that channel.

Second there is a canonical covariance option in the pull-down menu. This allows you to more carefully explore the correlation structure within and between subgroups of channels. You get to this via the pull down menu called "Plot Options". A set of pushbuttons listing all components in the array pops up, and you use this to pick the components in "group 1"; the remaining channels are assigned to group 2. The program then plots eigenvalues for each group separately, plus canonical coherences and covariances between the two groups. If there is coherent noise present which only occurs at some sites, or which only occurs in E components for example, this option may help you figure out which sites/components are not contaminated.

Under "Plot Options" there are also an option for plotting Signal Power, Noise Power, and Signal-to-noise ratios for all channels. The plots produced by clicking this button are pretty much self explanatory.

2.3 Plotting Array Transfer Functions : Pw_plot

This script reads in results output in the binary array TF file **array_name.Pw** output by **multmtrn**, and plots any chosen interstation and/or inter-component transfer functions desired, in any chosen coordinate system. Source code is in **matlab/PW**. After typing **Pw_plot** at the matlab prompt, a dialogue box opens for choosing the Pw-file to plot. Another dialogue box opens which controls the choice of transfer function, reference channels, and other plotting options. The program is fairly general, and this might cause problems in some cases. If you have data from several stations taken in different coordinate systems, you can pretty easily plot something that is not what you think it is. Read the instructions carefully.

There are essentially five separate things that you have to choose for plotting components of the general array transfer function: (1) Pairings of data channels for rotations of coordinate systems; (2) The rotation angle for the coordinate system; (3) The reference channels; (4) the polarization; and (5) the predicted components to plot. Also you can choose to plot results on linear or log scales, and as amplitude and phase or real and imaginary parts.

- (1) *Pairings of data channels for coordinate rotations.* Horizontal EM fields are generally collected in pairs, which together define a vector. The notion of coordinate rotations only makes sense for these vectors. In particular vertical magnetic components H_z are not part of any vector that is normally rotated in induction studies (of course H_z is part of a 3-D vector, but we generally only rotate coordinates about the vertical axis, so we only worry about rotations in the plane). Often, all components except possibly H_z will be paired in a natural way. But this will not always be the case: components may be missing due to instrument failure, or data may be collected in a configuration (e.g., EM profiling) with most E-dipoles oriented along a survey and only a few (or at least fewer) oriented perpendicular to the profile.

Pw_plot allows for channels to be paired up for this more general case. The program chooses "reasonable" default pairings, but these can be changed by editing in the dialogue box. Also, some components can be left unpaired (as individual channels). Again, the program chooses defaults for this (basically the H_z channels, or the E channels in case all are oriented the same direction), but these can be changed. The distinction between paired and single channels is significant in many cases. Paired channels will all be rotated into a common coordinate system defined by the rotation angle. Single components are not affected by rotations. Note that the program allows a single component to be used in multiple pairs. This is useful for rotating components collected in a profiling mode, where the numbers of E_x and E_y channels may be different.

The dialogue box contains two columns. Channels on the same line are paired. Channels on a line by themselves are not paired. The pair of channels in the first line will be used for the reference (see below). Note that the reference line has to have two channels for the plotting routine to work properly. The program starts by pairing all x and y channels at a station with the nearest (in the list of channels) channel of the same type (E or H). The names of the default pairings and single

channels are given in the text fields next to check boxes used for picking channels to plot. Each channel name has three parts: a channel type (H_x , E_y , etc), a site ID, and the orientation of the channel. Note that in some cases a single channel will be included in several pairs by default. Every channel is listed at least once; if a channel is not in any pairs, it is listed as a single channel. At present any channel included in any pair cannot be listed as a single channel. Each channel also has a number (the number in the list of components for the full array). You modify the pairings by editing the number boxes. There are basically three things you can do. (a) Delete a pair. To do this delete the number from one of the channels in the pair. This moves both channels into the list of single channels (unless the channel is listed in another pair). (b) change a pair by editing one of the number boxes, and (c) add a pair by putting the number of a channel into the right hand edit box next to the single channel you want to pair the channel with.

- (2) *The rotation angle* is changed by editing the *Rotation Angle* box. When plotting all paired channels are rotated into a right handed coordinate system (with the z -axis pointed down) with the x -axis pointed in the rotation angle direction, and the y axis 90 degrees clockwise. Note that channel pairs do not have to correspond to orthogonal directions to start with, but after rotation (and before plotting) everything is converted to a standard set of orthogonal coordinates.
- (3) *The reference channels* are the first rotation pair, at the top of the channel listing. To plot anything there has to be at least one rotation pair to serve as a reference. A single channel cannot be the reference.
- (4) *The polarization* is chosen by checking one of the boxes on the top line. Only one box can be checked. Checking the left hand box corresponds to the x -polarization. That is, this corresponds to the case where the x component of the reference vector (defined by the reference channels, but after rotation) is of unit magnitude with zero phase, while the y component is zero. Clicking the right hand box gives the y -polarization, again in the *rotated coordinates*. **NOTE:** Both reference channels figure in the definition of the polarization. **NOTE:** Everything is in rotated coordinates. If the first reference channel (in the left column) has an orientation of 270 degrees, the second reference channel has an orientation of 0 degrees, and the default x -axis orientation of zero degrees is used, clicking the left box corresponds to a unit source linearly polarized (at the reference site) with a direction of zero degrees. The right box gives a polarization pointing with a direction of 90 degrees clockwise. The original measurement coordinate definitions of x and y are not used when defining the polarization (but you could choose a rotation angle agreeing with the x -axis of the reference site to get TFs relative to measurement coordinate components.)
- (5) *Channels to plot* are chosen by clicking other check boxes. Any number of channels can be checked, but some may not be reasonable to plot (at least on the same scale). Again, the situation with the rotation pairs could be a bit tricky. The left hand box is the x -component of the *rotated* pair, not the original measured component (whose name is still displayed next to the check-box!) The warnings and explanations for

the polarization apply here also. *In summary: Individual channels can lose their identity after pairing and rotation. For plotting purposes the left box is always x, the right y, not the individual channels listed in the left column. The direction of the x-axis is determined by the rotation angle edit field. Unpaired channels keep their identity.*

After choosing TFs to plot, click **plot** to make the figure. By default, TFs are plotted as amplitude and phase, with amplitudes displayed on a log-linear plot. The display can be changed to log-log, or to real and imaginary parts of the TF using the pop-up menu in the dialogue box.

2.4 Adding TF curves to an existing plot

Pw_plot will plot as many TF components as you check off, but all of these have to be relative to a fixed polarization of a fixed rotation of a fixed pair of reference channels. To add TF components corresponding to different polarizations or reference channels (e.g., to plot H_z for both polarizations on the same figure), you can use the "Add TF" button. This can only be used after you have already plotted some TFs. Just pick reference channels, rotations, and predicted channels for the TF components you want to add, and click this button. A new figure will be plotted with both the existing TF components, and the new ones you have selected. You can add to the figure as many times as you want. The chosen reference channels and predicted channels are labeled on the right hand side of the figure.

3 M-files for reading tranmt and multmtrn output files

For the most important files output by **multmtrn** there are matlab M-file functions which read the files, and return generally reasonably named arrays containing the variables and header information stored in the files. These files can be found in `matlab/IN`. In some cases there are two M-files for each output file: one to open the file and read the header (given basic parameters like number of stations, number of channels in each station, number of frequency bands, etc.), and a second M-file for reading data from a single band. When there is a routine for reading the header, this routine usually also sets up some arrays which are used to make reading of individual bands simpler. The header routines must thus always be called first.

Note that these input functions are used by higher level routines (e.g., **apresplt.m** which is used for plotting apparent resistivities and phases). Also note that these routines all have comment headers for use with the matlab online help facility. Finally, there are no M-files for reading some rarely used output files.

3.1 station_name.z*

These files (one for each "station" in the array) contain local transfer function information, plus all error covariance matrices, channel orientations, etc., needed to calculate error bars

for any transfer function component in any coordinate system. Files with the same format result from single station and remote reference processing by **tranmt**. See the "Z-files" documentation in **doc/PS/Z_files.ps** for more details. This reading routine is used by **apresplt**, and forms the basis for matlab scripts used for assembling and plotting pseudo-sections.

Z_in.m : Reads in one Z-file

Usage: [z,sig_s,sig_e,periods,ndf,stcde,...
orient,nch,nche,nbt] = Z_in(cfile);

Input: cfile = Z--file name/path

Returns: nch = total # of channels ;
nche = nch-2 = # of predicted channels ;
nbt = # of bands
z(2,nche*nbt) = complex TFs
(NOTE: First two channels are always the "predictors")
(NOTE: Z(1,1:nche) corresponds to Hx sources for first band,
Z(2,1:nche) is Hy for first band,
Z(1,nche+1:2*nche) Hx sources for second band,
Z(2,nche+1:2*nche) is Hy for second band, etc.)
sig_s(2,2*nbt) = complex inverse signal covariance
sig_e(nche,nche*nbt) = complex residual error covariance
stdec(3) = station coordinates, declination
periods(nbt) = periods in seconds
orient(2,nch) = orientation (deg E of geomag N) for each ch

3.2 array_name.Pw

The array_name.Pw file contains full plane wave array transfer functions, with information needed to calculate array bars for transfer functions relative to any fixed reference. Reading routines are **Pw_hd** to read the header, and **Pw_in** to read in TFs and error covariances for a specified band.

Pw_hd.m : opens file, reads in header

Usage: [fid,recl,nbt,nt,nsta,nsig,nch,ih,stcor,decl,...
chid,csta,sta,orient] = Pw_hd(cfile);

Input: cfile = file name

Returns: fid = file id
recl = record length of direct access file
nbt,nt,nsta,nsig = # of bands, components, stations, evects
nch(nsta),ih(nsta+1),stcor(2,nsta),decl(nsta),sta(nsta),
orient(nsta) = the usual

\end{verbatim}

{\bf Pw_in.m} : reads in array TFs from Pw* file for one band

```
\small
```

```
\begin{verbatim}
```

```
Usage: [period,nf,tf,xxinv,cov] = Pw_in(fid,recl,ib);
```

```
Input:  fid = file id for input *.Pw file
        recl(2) = array of header, data record lengths returned
            by Pw_hd
        ib = frequency band desired

Returns: period, nf period, # of data points used for estimate
        tf(2,nt) array TF
        xxinv(2,2) inverse signal power matrix
        cov(nt,nt) complex Hermitian residual covariance (full matrix)
```

3.3 array_name.S0

The **array_name.S0** file contains the spectral density matrices, plus estimates of incoherent noise scales and all information about station locations/names, channel names, orientations, etc. Most of what is in the other *array* output files can be reconstructed from what is in this binary file. In the long run, only this file will survive, since all of the further computations done inside **multmtrn** can be quickly done in matlab! Reading routines are **uev_init** to read the header and initialize for the main reading routine, and **sdm_in** to read in SDMs and incoherent noise variance estimates for a specified band.

sdm_init.m : Initializes S0-file so SDMs, etc. can be read in

```
Usage: [fid_uev,irecl,nbt,nt,nsta,nsig,nch,ih,...
        stcor,decl,sta,chid,csta,orient,periods] = sdm_init(cfile);
```

```
Input:  cfile = file name
```

```
Returns: fid_uev = file id
        irecl = record length
        nbt,nt,nsta,nsig = # of bands, components, stations, evecs
        nch(nsta),ih(nsta+1),stcor(2,nsta),decl(nsta),sta(nsta),
        orient(nsta) = the usual
        periods(nbt) = periods
```

sdm_in.m : Reads in SDM for a single band

```
Usage: [period,nf,var,S] = sdm_in(fid_uev,nt,ib,irecl)
```

```
Input:  fid_uev = uev file id
        nt, = number of components
        ib = period band sought
        irecl = band record length
```

```
Returns: period = actual period
        nf = # of data vectors
        var = error variances for band ib
        S = SDM for band ib
```

3.4 SNarray_name

The file **SN_array_name** contains signal and noise power spectra. This information can in theory be read from the **array_name.S0** file, but it is easier to read from this simple ASCII file.

sn_in.m : reads in a SN_* file output by multmtrn

Usage: [ndf,E,S,NI] = sn_in(cfile);

Input: cfile = name for SN***** file to read

Returns: ndf(nbt) = # of data vectors used in each band
E(nt+1,nbt) = eigenvalues in noise units
S(nt+1,nbt) = signal power array
NI(nt+1,nbt) = incoherent power array

4 Some General Utility Functions

These are in subdirectory **matlab/UTIL**. Routines which are pretty specific to one type of plotting or analysis task are included in the directories for the higher level script (e.g., routines which only make sense for MT impedances are kept in **ZPLT**, and are described under the section on **apresplt**.

4.0.1 get_mode.m

This routine extracts elements of an array of the form that the transfer functions are stored in after reading by **Z_in.m** (i.e., $Z(2,nche * nbt)$, where *nche* is the number of predicted channels, and *nbt* the number of frequency bands. Returns the portion of the array corresponding to the mode defined by *ixy,icomp*:

- for H_y mode *ixy* = 2, *icomp* = tm dipole #s (= 2 for 5 component data)
- for H_x mode *ixy* = 1, *icomp* = te dipole #s (= 3 for 5 component data)

Here 5 component data refers to case of 3 channels predicted (by H_x and H_y), in the order H_z, E_x, E_y . Call this routine after any desired rotation. The routine can be called after conversion to *rho, ph, rho_se, ph_se* to extract appropriate elements of each of these arrays.

Usage: [mode] = get_mode(rho,ixy,nbt,icomp);

4.0.2 pol_ell.m

This function plots polarization ellipse centered at $(x(n), y(n))$ corresponding to the complex vectors $(dxr(n) + i * dxi(n), dyr(n) + i * dyi(n))$; $n = 1 : N$ *clr* is the color of the line used for the polarization ellipse. The routine calls ellipse.m to compute the ellipse. Scaling of the complex vector into the (x,y) space must be done before calling this routine

Usage: pol_ell(x,y,dxr,dyr,dxi,dyi,clr)

4.0.3 ellipse

This function constructs a curve for plotting the polarization ellipse centered at (x, y) corresponding to the complex vector $(u0, v0)$.

Usage: `[cuv] = ellipse(u0,v0,x,y)`

4.1 M-files in ZPLT

A number of the M-files used by **apresplt** will be useful for other applications involving impedance matrices or apparent resistivities and phases. These routines are in **ZPLT**.

4.1.1 z_to_imp.m

This function translates input from the general Z-file format to one or more impedance matrices with signal and noise covariance matrices necessary for full error computation in any coordinate system. In addition to arrays read in by **Z_in** the routine requires as input an array *ixy*(2, # of impedance matrices) which gives component numbers (first for Ex, second for Ey) to be used for each impedance matrix to be extracted. With a profiling setup with more dipoles along strike (x) than across (y), there may be some y components used for more than one impedance matrix. The routine also requires the array of channel orientations *orient*(*nch*) which gives orientations for *all* channels, including for the local reference (H) channels. On output the 2x2 impedances and all error covariance matrices are expressed in the coordinate system use to define the channel orientations—i.e., if orientations are expressed in geographic coordinates, impedances output by the routine are as well. The output arrays are all 4times $N_{imp}N_b$ where N_{imp} and N_b are the number of impedances extracted (one for a typical single site, but there could be more for profiling data), and the number of frequency bands. Results for all impedances for a single band are stored together. The order of the impedance elements in the 4 rows of Z_{2x2} is Zxx, Zxy, Zyx, Zyy, and similarly for the signal (*SIG_S*) and residual (*SIG_E*) covariances.

Usage: `[Z2x2,SIG_S,SIG_E] = z_to_imp(Z,Sig_e,Sig_s,Nche,ixy,orient);`

4.1.2 rot_z.m

This routine rotates impedances, signal, and residual covariances into a new coordinate system with the *x*-axis rotated θ degrees (positive θ is clockwise).

Usage: `[Z2x2R,SIG_SR,SIG_ER]=rot_z(Z2x2,SIG_S,SIG_E,theta)`

4.1.3 ap_res.m

This routine converts an array of impedances and error covariance matrices as read in by **Z_in.m** into apparent resistivity and phase, with error bars. Note that all elements of the TF matrix (including those corresponding to H_z TFs if appropriate) are converted to apparent resistivity. A subsequent call to routine **get_mode** is required to extract those elements of the converted arrays *rho*, *rho_se*, *ph*, *ph_se* which correspond to the appropriate off-diagonal elements.

Usage: [rho,rho_se,ph,ph_se] = ap_res(Z,sig_s,sig_e,periods) ;

Z = array of impedances (from Z-file)
sig_s = inverse signal covariance matrix (from Z-file)
sig_e = residual covariance matrix (from Z-file)
periods = array of periods (sec)

4.1.4 imp_ap.m

This routine computes app. res., phase, errors, given imped., cov. from *2times2* impedances and covariance matrices extracted from Z-files by **z_to_imp.m**.

USAGE: [ryx,rx,py,px,ryx_se,rx_se,pyx_se,px_se] =
imp_ap(Z2x2,SIG_S,SIG_E,periods);

INPUT:

Z2x2(4,:) = array of 2x2 impedance matrices
SIG_S(4,:) = inverse signal covariance matrix (2 H)
SIG_E(4,:) = residual covariance matrix (2 E)
periods = array of periods (sec)

4.1.5 pltrhom.m

This routine plots a series of ρ_a and ϕ curves, with error bars on a log-log scale for ρ_a and log-linear for ϕ . This version is a variant on **plot_rho.m** that allows for all arrays to be divided into a series of *NBT* bands, so multiple bands can be plotted with different symbols used for adjacent bands. Before calling this routine **set_fig** and **set_lims** must be called.

Usage: [rho_axes,ph_axes] = ...
pltrhom(NBT,pltind,periods,rho,rho_err,ph,ph_err,lims,c_title,hfig)

4.1.6 set_fig.m

This routine sets up apparent resistivity and phase figures, given plotting limits for ρ_a . Returns figure handle. Also sets up symbol styles, colors, etc. which are stored in global variables and then used by **pltrhom**. Call with a second argument (plot number) to make multiple plots—e.g., one for TM curves and one for TE curves for profiling data. This uses the specified plotting limits to determine figure dimensions so that reasonable MT aspect ratios are maintained.

Usage: [hfig] = set_fig(lims)
[hfig] = set_fig(lims,pltnum)

4.1.7 set_lims

This routine determines plotting limits for input to **set_fig**. First looks for a file called **limits.mat** in the current directory. If this is not found, and the routine is called with three arguments, the range of values found in array *rho* are used to set the plotting limits.

Otherwise some very broad default limits (which may be inappropriate in many cases) are used.

```
Usage: [lims,orient] = set_lims(dir,periods)
       [lims,orient] = set_lims(dir,periods,rho)
```

4.1.8 ecomp.m

This routine sorts out and pairs off E_x , E_y components for construction of conventional impedance tensors. The routine also returns a character string called DipoleSetup, which can take the value 'MT' for 2 E-channels, 'TEMAP' for tensor profiling data (i.e., some cross-profile dipoles), or 'EMAP' for data with no cross-profile dipoles.

```
Usage: [xy,yx,hz,xypairs,DipoleSetup] = ecomp(nche,chid);
```

4.2 M-files in SDM

Some of the M-files used by **sdm_plot** might be useful for other applications. Here are some notes about a few of these. These routines are in **SDM**.

4.2.1 ch_pair

This routine pairs off E and H channels at a single site following simple default rules. For each station (i.e., FC file grouping) the first H_x is sought and the first H_y . If both are found a pair is created (all other magnetic channels in the "station" are ignored). The same procedure is followed for E_x and E_y . Indices in the total channel list are returned for those component pairs found for E and/or H. It is OK to have one sort of channel, but not the other. Thus 0-2 rotatable channel pairs identified for each station. Other sorts of pairings which might be desired will require other routines/direct user specification.

```
Usage: [Hp,Ep,HZ] = ch_pair(nch,chid);
Hp/Ep give pairs of H/E channels, HZ gives first HZ at
each station if any; for Hp/Ep(HZ) column 3(2) gives
the corresponding station #
```

4.2.2 u_pair

This routine uses the indices of pairs of H and E data channels specified in arrays Hp/Ep/HZ, to construct complex vectors, which are then converted into a common standard coordinate system (using array orient).

```
Usage : function [Uh,Ue,Uz] = u_pair(u,Hp,Ep,HZ,...
                                orient,decl,stcor,period,rho_ref)
```

4.3 M-files in PW

Some of the M-files used by **Pw_plot** might be useful for other applications. Here are some notes about a few of these. These routines are in **PW**.

4.3.1 pwstruct.m

This routine loads in contents of the Pw-file, puts the header information into the data structure **pwhd**, and the array TFs and error covariances into the data structure **pw**. The structure **pw** contains data from all frequency bands. The fields defined for the structures are:

```
pw = struct( 'T',period,  
'nf',nf,  
'tf',tf,  
'xxinv',xxinv,  
'cov',cov);  
pwhd = struct( 'nbt',nbt,  
'nt',nt,  
'nsta',nsta,  
'nsig',nsig,  
'nch',nch,  
'ih',ih,  
'stcor',stcor,  
'decl',decl,  
'chid',chid,  
'sta',sta,  
'orient',orient,  
'ch_name',ch_name);
```

Usage: [pw,pwhd] = pwstruct(cfile)

4.3.2 rotatePw.m

This routine rotates pairs of data channels specified in array **rot_ch**(nrot,2), modifying the transfer function and residual covariance fields **Pw.tf** and **Pw.cov** of data structure **Pw**. Channels listed in **sing_ch** are also included in the output structure without rotation. Channels may be used for multiple rotations, so the total number of channels output may exceed the number of channels in the input structure. (But note that in this case the residual covariance matrix will be singular) Uses orientations in **Pwhd.orient**, and angle **theta** for new coordinate axis. Output is another array TF structure **Pwrot**, with the same number of frequency bands as the input structure **Pw**.

Usage : [Pwrot] = rotatePw(Pw,Pwhd,rot_ch,sing_ch,theta);

4.3.3 pwrfsite.m

This routine uses the two array TF vectors in **Pw.tf**(2,nt) to compute TFs for all components relative to two specified reference components specified in array **ref**(2). Also outputs one standard error for these linear combinations using full error covariance info as input in **Pw.xxinv**, **Pw.cov**

Usage: [v,sig_v] = pwrfsite(Pw,ref);

Errors Bars for Transfer Function Elements in Z-files

Gary D. Egbert*

February 27, 1998

1 File Format

The Z-files contain all information needed to compute standard transfer functions (e.g., impedances), with error bars in any coordinate system. Here is an overview of the format, followed by an artificial example, with some annotation off to the side. The basic idea is that there are NCH channels, with the first two used as the "local reference" i.e., these are the input or predictor channels (classically the local horizontal magnetics), and the remaining $NCH-2$ are the output or predicted channels (the electrics, and/or vertical magnetics). Note that there might also have been another pair of channels (or a whole array) used as a remote reference. These possible other channels are not referred to explicitly in this file (but they were used to compute the contents of the file). A file of this same format can in principle be produced from single station, standard remote reference, or the multiple station program. This effects how the contents of this file was created, but not any subsequent calculations using this file.

In overview, the file is ASCII, with a short header block which identifies the NCH channels. There are then a series of $NBANDS$ blocks, one for each period for which an estimate has been computed. Each period block contains three complex arrays:

1) Z : the transfer function (TF) array. This array is $NCH-2$ rows by 2 columns. For $NCH = 4$, with two reference channels H_x and H_y , and two predicted channels E_x and E_y , Z is just the impedance tensor.

2) S : the "inverse signal covariance" array. This is a 2×2 Hermitian matrix. Only the 3 elements corresponding to the part on and below the diagonal are actually in the file. These elements are given in the order S_{11} , S_{21} , S_{22} . The missing element satisfies $S_{12} = S_{21}^*$, where the superscript asterisk denotes the complex conjugate. Note that in the case of a single station impedance estimate S is just the inverse of the H cross power matrix. The exact form is slightly different for the case of remote reference or array results. This matrix is needed for the error calculation. (Actually only the diagonal elements, which are real, are needed unless you rotate the coordinate systems).

3) N the residual covariance matrix. This is an $(NCH - 2) \times (NCH - 2)$ Hermitian matrix, output in the same symmetric form as S . This gives the covariance of the residuals for all predicted channels. Again, only the diagonals (also real) of this matrix are needed

---

*College of Oceanic and Atmospheric Sciences, Oregon State University, Corvallis; egbert@oce.orst.edu

for error calculations in the "default" coordinate system, but other parts of the matrix will be used for a correct treatment of coordinate changes/rotations.

(Mike: Actually there might be a slight difference in the header block format in the version you have; this is for the most recent version, and I'm not sure exactly what Clark is using).

```
TRANSFER FUNCTIONS IN MEASUREMENT COORDINATES    <==== line 1 of file
***** WITH FULL ERROR COVARAINCE*****
```

```
S2                                     <===== some sort of "station" id
coordinate    49.28    102.91 declination    0.00    <=== station coordinates
number of channels    5    number of frequencies    26    <=== NCH, NBANDS
orientations and tilts of each channel
    5    0.00    0.00 S2H    Hx    <=== for each channel in this "station":
    6    90.00    0.00 S2H    Hy    (1) channel number (this came from multmtrn
    7    0.00    0.00 S2H    Hz    and so the #s aren't 1,2 ... ;
    8    0.00    0.00 S2E    Ex    (2) orientation (deg. E of N); (3) tilt ;
    9    90.00    0.00 S2E    Ey    (4) Data logger ID (5) channel type

period :    4.65455    decimation level    1    freq. band from    25 to    30
number of data point    2496 sampling freq.    1.000 Hz    <=== info about 1 band
Transfer Functions
    0.2498E+00 -0.2049E-03 -0.9341E-04  0.2517E+00
    -0.6246E-02 -0.5245E-01 -0.7291E+01 -0.7318E+01    <=== Z
    0.7292E+01  0.7346E+01 -0.3806E-01  0.5754E-02
Inverse Coherent Signal Power Matrix
    0.2947E-07  0.5753E-16    <=== S
    -0.1575E-09  0.1391E-09  0.2895E-07  0.2386E-15
Residual Covariance
    0.3198E+02  0.0000E+00
    0.2252E+03 -0.2185E+03  0.2660E+05  0.0000E+00    <=== N
    0.2424E+03  0.2418E+03  0.4577E+03  0.3710E+03  0.2781E+05  0.0000E+00
```

The block given above for one period should be pretty much self explanatory. Note that the last two rows of the TF matrix are the local impedance tensor.

2 Error Calculation

First I just give the formula for calculating errors in the TF given in the files (i.e., in the "default" measurement coordinate system). Next, I'll give formulas for transforming the matrices **Z**, **S**, **N** into a different coordinate system (not necessarily by rotation). The initial formulas for TF error in the measurement coordinate system can then be applied to the transformed matrices. Finally, linear error propagation is applied to give the standard error estimates for ρ_a and ϕ computed from the off-diagonal elements of the impedance.

2.1 Errors In Transfer Functions

The error covariance for the elements of the transfer function matrix $\mathbf{Z}$ is given by:

$$\text{Cov}[Z_{ij}Z_{i'j'}] = N_{ii'}S_{jj'} \quad j, j' = 1, 2, \quad i, i' = 1, NCH - 2. \quad (1)$$

You will normally only care about the variances (i.e., the case where $i = i'$ and $j = j'$). In this case you would use only the diagonal elements of $\mathbf{S}$ and $\mathbf{N}$. In the following I refer to these variances as

$$\sigma_{ij}^2 = \text{Var}[Z_{ij}] = \text{Cov}[Z_{ij}Z_{ij}] = N_{ii}S_{jj}. \quad (2)$$

For example, the impedance element $Z_{xy}(= E_x/H_y)$ in the above example is element (2,2) (row = $i = 2$, column = $j = 2$) in the TF matrix $\mathbf{Z}$. The error variance is obtained from the product of the second diagonal element of the inverse coherent signal power matrix S_{22} , and the second diagonal element of the residual covariance N_{22} . The other off-diagonal impedance element Z_{yx} corresponds to $i = 3$ and $j = 1$, and the variance is $\sigma_{31} = N_{33}S_{11}$. Note that this gives the variances of the complex transfer functions; variances of real and imaginary parts separately are each one half of the complex variance given by (1).

2.2 Transformation Of Transfer Functions and Errors

The transformation of error covariance can be computed for any linear transformation of the predicted and predictor channels. Here I just give expressions for the most standard rotations. Denote by $\theta_1, \theta_2, \dots, \theta_{NCH}$ the channel orientations (these are given in the header block of the $\mathbf{Z}_-$ file). Let θ be the desired rotation of the x -axis, relative to the same reference direction used to define the channel orientations (e.g., geographic or geomagnetic north). Note that the sort of coordinate changes we focus on here implicitly involve pairs of channels (the two reference magnetics; a pair of electric channels). Vertical magnetics are not rotated (well ... we *could* get into allowing for tilt ...), and when there are multiple electrics, it will be necessary to identify pairs of channels to transform together. I thus describe transformation of one pair of channels at a time, say channels l, m . Form the matrix

$$\mathbf{U}_{lm} = \begin{bmatrix} \cos(\theta_l - \theta) & \sin(\theta_l - \theta) \\ \cos(\theta_m - \theta) & \sin(\theta_m - \theta) \end{bmatrix}^{-1} \quad (3)$$

Note that if you form the 2-vector $\mathbf{x}$ from the (l, m) pair of measured data channels, then $\mathbf{U}_{lm}\mathbf{x}$ gives the vector expressed in the new right-handed orthogonal coordinate system (with x -axis pointing in the direction θ degrees E of the reference direction). Note that in the "usual" MT case where there is one pair of reference channels H_x, H_y and one pair of predicted channels E_x, E_y , and both are expressed in the same orthogonal coordinate system, then the same matrix $\mathbf{U}_{lm}$ would be used for coordinate transformation of both pairs, and we would also have $\theta_m = \theta_l + 90$. In this case $\mathbf{U}_{lm}$ would reduce to the more familiar form for the impedance tensor rotation matrix. The formulas given here work for any orientations, including the case of non-orthogonal measurement component pairs.

First consider transformation of the predicting channels $l = 1, m = 2$ (normally these would be H_x, H_y). $\mathbf{Z}$ and $\mathbf{S}$ are effected by this part of the transformation. In the new coordinate system the matrices are:

$$\mathbf{Z}' = \mathbf{Z}\mathbf{U}_{12}^T \quad \mathbf{S}' = \mathbf{U}_{12}\mathbf{S}\mathbf{U}_{12}^T \quad (4)$$

The output residual covariance $\mathbf{N}$ of course remains unchanged by a transformation of only the input channels.

Next consider transformation of two of the output channels, $3 \leq l, m \leq NCH$. (Note the numbering convention: output channels start with 3, and go to NCH , for a total of $NCH - 2$). The simplest way to express the result in general is to define an $(NCH - 2) \times (NCH - 2)$ transformation matrix $\mathbf{V}_{lm}$ which rotates only channels l and m . For the example file above, where $NCH = 5$, the matrix for rotating the coordinate system for the pair of electric field channels (i.e., $l = 4, m = 5$), $\mathbf{V}_{45}$ would take the form

$$\mathbf{V}_{45} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_4 - \theta) & \cos(\theta_5 - \theta) \\ 0 & \sin(\theta_4 - \theta) & \sin(\theta_5 - \theta) \end{bmatrix}. \quad (5)$$

More generally the following pseudo-code defines $\mathbf{V}_{lm}$, assuming $l < m$:

$\mathbf{V}_{lm} = (NCH - 2) \times (NCH - 2)$ identity matrix

$\mathbf{V}_{lm}(l - 2, l - 2) = \cos(\theta_l - \theta)$

$\mathbf{V}_{lm}(m - 2, l - 2) = \sin(\theta_l - \theta)$

$\mathbf{V}_{lm}(l - 2, m - 2) = \cos(\theta_m - \theta)$

$\mathbf{V}_{lm}(m - 2, m - 2) = \sin(\theta_m - \theta)$

With $\mathbf{V}_{lm}$ thus defined the transformations of $\mathbf{Z}$ and $\mathbf{N}$ are:

$$\mathbf{Z}' = \mathbf{V}_{lm}\mathbf{Z} \quad \mathbf{N}' = \mathbf{V}_{lm}\mathbf{N}\mathbf{V}_{lm}^T \quad (6)$$

In general both input and output channels will be rotated, so both (4) and (6) will be used. In the 5 channel example given above the full transformation of all arrays is thus:

$$\mathbf{Z}' = \mathbf{V}_{45}\mathbf{Z}\mathbf{U}_{12}^T \quad \mathbf{S}' = \mathbf{U}_{12}\mathbf{S}\mathbf{U}_{12}^T \quad \mathbf{N}' = \mathbf{V}_{45}\mathbf{N}\mathbf{V}_{45}^T. \quad (7)$$

More generally there may be a series of electric field pairs, requiring that (6) be applied for each pair. Note that in this case a single matrix $\mathbf{V}$ can be derived which transforms all channel pairs, by starting with the $(NCH - 2) \times (NCH - 2)$ identity matrix and modifying the appropriate four elements of $\mathbf{V}$ for each pair l, m . Error (co)variances for the transformed impedance elements are then as given in (2) and (1), with $\mathbf{N}'$ and $\mathbf{S}'$ replacing $\mathbf{N}$ and $\mathbf{S}$.

3 Apparent Resistivities and Phases

After transforming all three arrays, apparent resistivities (ρ_a), phases (ϕ) and error bars ($\sigma_\rho; \sigma_\phi$) can be computed from the appropriate off-diagonal impedance elements (say Z_{ij}), the period T , and the associated error variance σ_{ij} given above. For completeness here are the expressions derived from linear propagation of errors, under the assumption that errors are small compared to the impedance.

$$\rho_a = \frac{T|Z_{ij}|^2}{5}$$

$$\sigma_\rho = \left[\frac{(2T\rho\sigma_{ij}^2)}{5} \right]^{1/2}$$

$$\phi = \frac{180}{\pi} \arctan[\Im(Z_{ij})/\Re(Z_{ij})]$$

$$\sigma_\phi = \frac{180}{\pi|Z_{ij}|} \left[\frac{\sigma_{ij}^2}{2} \right]^{1/2}$$