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

Gary D. Egbert & Markus Eisel*

March 24, 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 is 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

    MMT      ==>   empty directory for output from multmtrn

    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, MT and MMT directories
                 as each testing step is completed.
```

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

    EMI-MT1 ==>  test data for EMI MT-1 reformatting (with similar
  (subdirectories as in test)

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 Portability

The code has been developed and tested primarily on a UNIX sun workstation running Solaris. It has also been tested on IBM workstations running AIX, and on PCs running Windows 95 (Watcom Fortran compiler) and linux. With different systems/compilers you may have some minor problems, but probably nothing serious. **rfemi**) might be more trouble on a PC, since this links c and fortran routines, which will probably require compatability between c and fortran compilers.

1.5 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). If all of the Makefiles are OK for your system (e.g., if you are compiling on a SUN running SOLARIS 2.x everything should be OK) you can just execute the simple shell script INSTALL in subdirectory test. This will make and install executables in test/bin. To compile **multmtrn** and/or **rfemi** uncomment the appropriate lines in this file–as it is now, only the "standard" programs **rfasc**. ,**dnff**, and **tranmt** are compiled and installed by this script. Note that in many cases some of the programs **rfemi** and **multmtrn** will not be successfully made by just typing INSTALL, since: (1) **multmtrn** needs to link with some routines from a **lapack** library. These are assumed by default to be in /usr/local/lib/liblack.a and /usr/local/lib/libblas.a. If these libraries are not available on your system, you will need to make and install them following instructions in **MMT**. (2) **rfemi** links with a c routine. On different systems

there are different conventions for doing this.

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 **datsz.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 a 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 Making and Testing rfemi

To make and test **rfemi**, follow the instructions in the **README** file in **test/EMI-MT1**. A small test data set in raw EMI-MT1 format is provided in this directory, along with all files needed to processes this data. **rfemi** is compiled as for **rfasc**, and the program produces identical output binary data files. Note that this program also reads the EMI MT-1 data file headers and automatically generates system paramter (***.sp**) files.

2.4 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.

2.5 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 *datsz.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,Hz,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 **datsz.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**.

nwmx

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). The program stops FFTing once the number of sets exceeds *nsmax* and issues a warning.

nbadmx

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. This parameter is actually set in another include file *../include/nchmx.inc* which is included in *params2.inc* and also in the other programs of the **EMTF** package.

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!). At present, only the number of decimation levels and the prewhitening option for the first channel for each level is read. Thus you do not need a different pwset.cfg file for different numbers of data channels. 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).

NOTE: To repeat, only the first number from each line in this file is now read; pwset.cfg as reproduced above could be used with any number of data channels.

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.

Note: Several parameters (orientation, decl, coordinates, etc.) are passed via the FC files to the transfer function programs **tranmt** and **multmtrn**. They are essential for proper output of results and subsequent plotting. It is recommended to carefully set up the **SP** files atthe beginning of te processing.

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 up to 6 characters long. These IDs are passed on to intermediate and final results files. Some matlab programs for plotting results expect the first two characters of the channel IDs to be Hx, Hy, Hz, Ex, or Ey to allow the programs to make default choices about modes to plot etc. It is thus best to use the first two characters to identify field type and nominal orientation, and any subsequent character (up to 4 additional) for additional identification if desired. 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 form 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 corrections 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 (Aa - Zz, &$%^#, etc.) 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. **resptbl** 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

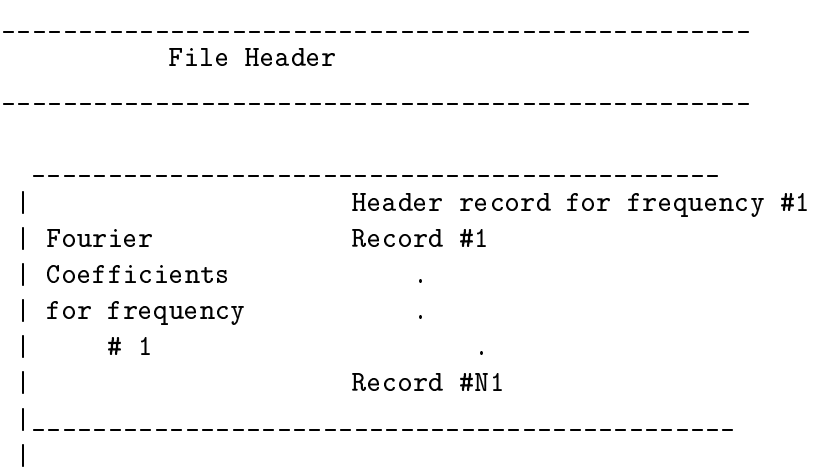
(2) running **dnff -a** directly on **test1.asc**.

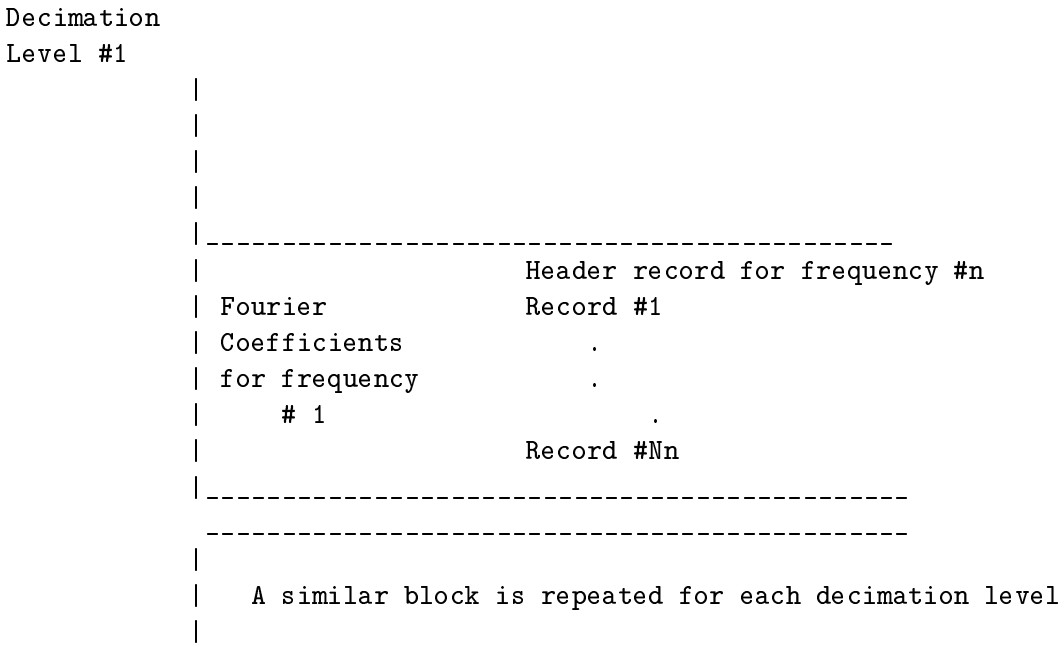
The modifications to **dnff** to allow direct reading of these simple ASCII files is a simple example of how this program can be adapted to directly read other file formats. This is discussed further below in section 4.6.

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:





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 **asc_rec** was added (in **inpu_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 **inpu_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 Command line options

there are three command line options which can be used with **tranmt**:

- **-s**⟨**ref-file**⟩ Tells **tranmt** to read a new order of predicting, predicted (and remote reference) channels from file *ref-file*.
- **-x** Output the optional spectral density matrix file.
- **-p** Print the first/last set number for each FC file and stop. This is useful for using the **SETS** option described below.
- **-S**⟨**sets_to_process**⟩ This option tells the program to only process certain subsets out of the total number of sets. This is aquivalent to process only a segment of the total time series. The sets to e processed are specified by a first/last set number. The progrma allows to specify several of such "windows". The syntax of the command line option **-S** is:
  -SN,m11,m12,...mn1,mn2,...mN1,mN2
  where N (≤ 10) is the number of windows, mn1,mn2 are first and last set number of window n. Use the **-p** option to find out which set numbers are covered by the data to process. Using this option produces extra output as in the example below.

```
*********************************************

                SET-WINDOWS FOR PROCESSING
SET-WINDOW:   1/1    1/2    2/1    2/2
DEZ-LEVEL
    1       16700 18000 19000 19500
```

```
2        4175  4500  4750  4875
3        1044  1125  1188  1219
4         261   282   297   305
5          66    71    75    77

*********************************************
```

5.3 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**.

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 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 neccessary 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**.

5.4 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**.

5.5 Configuration files

5.5.1 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.
```

5.5.2 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, cohp(1), cohp(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 paramters 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) *cohp* two real parameters used to determine a target number of degrees of freedom in the transfer function estimate via :

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

5.5.3 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:

```
28                              No. of bands
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.

5.5.4 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.

5.6 Output

Currently **tranmt** outputs two files: A "Z–file" containing local TFs and error bars and an optional file containing cross-spectra for all predicted and reference channels in a format compatible with the GEOTOOLS **ediwrite** program (see below for details). 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"**.

The cross-spectra files have the extension **sdm** (in the example it would be test1.sdm). This output is optional and invoked using the command line option "-x". As the format is fully compatible with the EDI-reformatting routine **ediwrite**, this option only supports 5

and 7 channel data, i. e. standard MT and MT plus remote reference setups. The spectral density matrix is rotated into a geographic north coordinate system, based on the sensor orientations and the declination. These values need to be set correctly before FFTing the time series with **dnff**. In addition to the output file **tranmt** prints out a line which can be used as a template for the *sites* file necessary to use **ediwrite**. **tranmt** only checks for the number of channels to be equal 5 or 7, but not for the sequence of channels (GEOTOOLS requires HX, HY, HZ, EX, EY (HXR, HYR)). (subroutines: sdmwrite, ccmat, rotsdm).

5.7 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., datsz.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, *Geophs. J. Int.*, **130**, 475-496, 1997.

8 Appendix: Lists of source code files

DNFF

afcor.f	autocor.f	badrec.f	bdrcsu.f	chdec.f	cldrft.f
cmove.f	dcfilt.f	dcimte.f	decset.f	demean.f	dnff.f
dtrnd.f	fcorsu.f	filtcor.f	freqout.f	frstdif.f	ft_subs.f
getsp.f	inpu_asc.f	inpu_bin.f	ltslv.f	mk_offst.f	mkset.f
out_pack.f	phs_shft.f	ptdist.f	pterst.f	resptbl.f	sort.f

TRANMT

bset.f	chdec.f	cmove.f	cohband.f	cohsrt.f	corchng.f
edfwts.f	fop.f	matmult.f	mcohc.f	mkfdir.f	mkhat.f
mkrec.f	mkrhat.f	modobs.f	movec.f	mult.f	pwrvf.f
rbstrn.f	rdcndwt.f	readfvg.f	reorder.f	rfhead.f	rotsdm.f
rtreref.f	rxspclev.f	savtf.f	sdmsort.f	sdmwrite.f	setup.f
sindex.f	sort.f	stack.f	tranmt.f	trlsrr.f	unstack.f
wrt_z.f	wt.f				

MULTMTRN

anfld.f	ap_res.f	canonic.f	cn_wt.f	corchng.f	extra.f
filtpc.f	geogcor.f	grad.f	ln_rbst.f	minpwr.f	mk_fd_rec.f
mk_list.f	mmt_mtrx.f	mmt_subs.f	multmtrn.f	n_rbst.f	pc_coeff.f
pc_out.f	prteig.f	rbstk2.f	rbstreg.f	readfvg.f	refproj.f
rr_rbst.f	rsp.f	sep_s_n.f	setup.f	sindex.f	timerr.f
var_adj.f	wrt_z.f	wrtx.f			