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 each 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 aprespl.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 $(1, 1), (2, 2), ...(nsta, nsta)$. 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, *rho_ref*. If the actual apparent resistivity has equals *rho_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 loose 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 labled 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 for first band,
                       Z(1,nche+1:2*nche) Hx sources for second band,
                       Z(2,nche+1:2*nche) is Hy for 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, evecs
           nch(nsta),ih(nsta+1),stcor(2,nsta),decl(nsta),sta(nsta),
           orient(nsta)  = the usual
\end{verbartim}

{\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 Hy mode ixy = 2, icomp = tm dipole #s (= 2 for 5 component data)
- for Hx 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 $4timesN_{imp}N_b$ 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 $Z2x2$ 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,rxy,pyx,pxy,ryx_se,rxy_se,pyx_se,pxy_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 phi curves, with error bars on a log-log scale for rho_a and log-linear for phi. 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 profilling 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 fucntion 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);
```