ModEM: User's Guide

Anna Kelbert (COAS/OSU)

December 22, 2011

1 What is ModEM?

Modular EM (**ModEM**) is a flexible electromagnetic modelling and inversion program written in Fortran 95. It is currently available to use with 2D and 3D MT problems. It can also be quite easily extended to do other things, but code modification is beyond the scope of this document (see Egbert et al., 2011). The program has a command-line interface and will work with most Fortran 90/95 compilers on most platforms.

2 Where do I get it?

ModEM 2D and 3D MT modelling and inversion codes are currently available for non-commercial and academic use subject to the attached license agreement, see COPYRIGHT. The latest stable version of ModEM can always be obtained from our Subversion repository:

```
http://mt.coas.oregonstate.edu/svn/ModEM/branches/stable
```

It is advisable that you obtain the code through the Subversion repository, to keep your version up-to-date with bug fixes and updates. Please refer to the README file for details on obtaining and updating the code. Please contact the authors to obtain a username and password to the repository. Alternatively, the authors may be willing to provide a packaged archive or an executable.

3 How do I run it?

The file structure of ModEM looks like this:

COPYRIGHT Please get familiar with the Copyright before using this code!

README Explains how to obtain, install and update the code.

doc/ Provides additional documentation, including this user guide.

examples/ Provides a careful set of examples that are known to work.

f90/ The code base, makefiles and configuration scripts.

matlab/ Auxiliary Matlab scripts for file conversion, and others.

Unless you edit the code substantially, you wouldn't need to use any configuration scripts. Please use the provided makefiles instead.

There are several makefile variants for both 2D and 3D MT. Please choose Makefile3d if you would like a serial version of the code, and Makefile3d.MPI for a parallel version. Then, copy your selection to a new makefile that you could easily edit, without conflicting with the configurations stored in Subversion control system, e.g.,

```
cd f90; cp Makefile3d Makefile
```

Then, please open the Makefile and make sure that the uncommented compiler is available on your system. Then, the `make` command will compile the 3D MT (or 2D MT) code for you (`Mod3DMT` and `Mod2DMT`, respectively).

Running these programs without command line arguments will print the following lines:

```
 Output information to files, and progress report to screen (default).
 Usage: Mod3DMT -[job] [args]

  [READ_WRITE]
   -R  rFile_Model rFile_Data [wFile_Model wFile_Data]
    Reads your input files and checks them for validity;
    optionally also writes them out
  [FORWARD]
   -F  rFile_Model rFile_Data wFile_Data [wFile_EMsoln rFile_fwdCtrl]
    Calculates the predicted data and saves the EM solution
  [COMPUTE_J]
   -J  rFile_Model rFile_Data wFile_Sens [rFile_fwdCtrl]
    Calculates and saves the full J(acobian)
  [MULT_BY_J]
   -M  rFile_Model rFile_dModel rFile_Data wFile_Data [rFile_fwdCtrl]
    Multiplies a model by J to create a data vector
  [MULT_BY_J_T]
   -T  rFile_Model rFile_Data wFile_dModel [rFile_fwdCtrl]
    Multiplies a data vector by J^T to create a model
  [INVERSE]
   -I NLCG rFile_Model rFile_Data [lambda eps]
    Here, lambda = the initial damping parameter for inversion
             eps = misfit tolerance for the forward solver
  OR
   -I NLCG rFile_Model rFile_Data [rFile_invCtrl rFile_fwdCtrl]
    Optionally, may also supply
        the model covariance configuration file   [rFile_Cov]
        the starting model parameter perturbation [rFile_dModel]
    Runs an inverse search to yield an inverse model at every iteration
  [TEST_COV]
   -C  rFile_Model wFile_Model [rFile_Cov]
    Applies the model covariance to produce a smooth model output
  [TEST_ADJ]
   -A  J rFile_Model rFile_dModel rFile_Data [wFile_Model wFile_Data]
    Tests the equality d^T J m = m^T J^T d for any model and data.
    Optionally, outputs J m and J^T d.
  [TEST_SENS]
   -S  rFile_Model rFile_dModel rFile_Data wFile_Data [wFile_Sens]
    Multiplies by the full Jacobian, row by row, to get d = J m.
    Compare to the output of [MULT_BY_J] to test [COMPUTE_J]

  Optional final argument -v [debug|full|regular|compact|result|none]
  indicates the desired level of output to screen and to files.
```

For an average user, options [READ_WRITE], [FORWARD] and [INVERSE] would be most directly applicable. Subdirectory `examples/` contains sample input files with instructions. These should be sufficient to test your installation. They should also provide a suitable template for your own applications.

At this time, we are not able to provide a thorough documentation. However, please refer to `doc/` subdirectory for further information about ModEM.

4 How do I get my data in the right format?

The data format used in ModEM is the so-called "list format". This format hasn't been widely used before, so it requires a detailed description.

The data file on input to ModEM contains one or more blocks of data. Each block corresponds to a single kind of data, and consists of an 8-line header followed by the data. Example header looks like:

```
# Cascadia data from XML files, error floor 5%, no topography
# Period(s) Code GG_Lat GG_Lon X(m) Y(m) Z(m) Component Real Imag Error
> Full_Impedance
> exp(+i\omega t)
> [mV/km]/[nT]
> 0.00
> 45.276 -119.634
> 10 109
```

The data are written in a list, one data component in a line, in the format described in the header. In this case, each line would contain a period, site code (up to 12 chars), geographic latitude and longitude, X/Y/Z location in meters, component code (e.g., ZXY), real and imaginary parts, and the error bar.

The purpose of the header and your options are described in detail below. I have also provided a set of examples in the `examples/` directory. The `matlab/ioAscii/` directory contains the read and write routines in Matlab (these are restricted to Off Diagonal Impedance, Full Impedance or Full Impedance plus Full Vertical Components file variants). There are also scripts to convert the older data files to the current ModEM format. Finally, the `examples/3D_MT/Cascadia/` directory has a Perl script that converts a WS data format to ModEM data format.

Line 1: # comment

Comment line, intended to describe your data; will be read and replicated in output responses. Max 100 chars.

Line 2: # comment

Comment line, intended to describe the data format; will be ignored by the code.

For complex impedances and vertical components the data format is:

```
Period(s) Code GG_Lat GG_Lon X(m) Y(m) Z(m) Component Real Imag Error
```

For apparent resistivity and phase the data format looks like:

```
Period(s) Code GG_Lat GG_Lon X(m) Y(m) Z(m) Component Value Error
```

For interstation transfer functions, information about the reference site is also required:

```
Period(s) Code GG_LatLon XYZ(m) Ref_Code Ref_LatLon Ref_XYZ(m) Component Real Imag Error
```

For 2D MT, the data format is identical, but the X coordinate is not used or stored in the program.

Line 3: > data_type

Keyword for the data type stored in this data block. For 3D MT:

Keyword	Allowed components	Real or complex
Full_Impedance	ZXX,ZXY,ZYX,ZYY	Complex
Off_Diagonal_Impedance	ZXY,ZYX	Complex
Full_Vertical_Components	TX,TY	Complex
Full_Interstation_TF	MXX,MXY,MYX,MYY	Complex
Off_Diagonal_Rho_Phase	RHOXY,PHSXY,RHOYX,PHSYX	Real

For 2D MT:

Keyword	Allowed components	Real or complex
TE_Impedance	TE	Complex
TM_Impedance	TM	Complex

Missing components are allowed. However, it is advisable to resort to the more restrictive data type whenever possible, as this will minimise internal storage and computations (e.g., use Off_Diagonal_Impedance in place of Full_Impedance, whenever possible).

If you would like to add a data type that is not described above, please get in touch with us directly. In many cases, it will be a trivial addition.

Line 4: > sign_convention

Plus or minus, written as, e.g., "exp($+i\omega t$)". We search for the minus sign in this line; if not found, assume a plus. The data are then converted to the sign convention used internally in the code (which is determined by the global variable `ISIGN`).

Line 5: > units

Units for this data type.
Internally in the code, the SI units for E/B are used for MT impedances. However, the user is given a choice as to the input units. The options are:

1. SI units for E/B: [V/m]/[T] (used in ModEM)
2. practical units for E/B: [mV/km]/[nT]
3. SI units for E/H: [V/m]/[A/m] = Ohm

The output data will be given in the same units as the input. The units supplied in the data file must of course match the data.
Dimensionless data (such as the vertical magnetic field components) are identified by empty "[]" units. Apparent resistivity and phase are loosely viewed as being dimensionless (this may change if they are separated into two separate data types).

Line 6: > orientation

Geographic orientation of all data components relative to geographic North. Normally, in a 3D MT application, all impedance are already rotated to geographic North. In this case, orientation should be set to zero. In a 2D MT application, non-zero data orientation would not be uncommon. This field is not used by code at present. It is mainly intended for the user, to facilitate keeping track of their data orientation. Normally, all data would be oriented in the same direction.

Line 7: > geographic_origin

Latitude and longitude for the (0, 0, 0) point in the data file. Not used in ModEM. Very useful for plotting, and to match the origin of the model to that of the data. Set to zero if unknown.

Line 8: > nperiods nsites

Maximum numbers of periods and sites in this data block. Needed for dynamic allocation of the arrays when the file is read. We suggest that you use these to keep track of your periods and sites, but they could also be set to a large value. Missing data are allowed, and if a period or site completely disappears from your file, things should still work.

Currently, if these numbers are smaller than required to read your data block, segmentation fault will occur. If unsure, use the command

```
    ./Mod3DMT -R model_input data_input model_output data_output
```

to validate your model and data files. This will read your files, store them in the internal dictionary, data and model structure, write them out again, and exit.

5 What is your model format?

For convenience of the users, we have employed the model format identical to the default used in Weerachai Siripunvaraporn's WSINV3D program. The only amendment, however, is that we have added an option to store the resistivity values in natural log format. To do so, we include a `LOGE` specifier in the header line, along with the model size. E.g., the line

```
48 46 34 0 LOGE
```

would indicate a model size $48 \times 46 \times 34$ written in natural log format. Linear resistivity is currently immediately converted to natural log on input to ModEM.

The origin in the model file must point to the (0, 0, 0) location in the data file. The model origin in metres and the model rotation are recorded, respectively, in the last two lines of the model file (which are optional). Model rotation is not currently supported.

If the origin is not specified, the middle point is used for the origin. It is the user's responsibility to ensure that the model and data match. For optimal accuracy of conversion between spherical and cartesian coordinates, we suggest that the user chooses an origin that is reasonably close to the center of the model. However, instead of using the middle point, it is usually best to choose one of the center site locations as the origin: then, the data file needn't be recreated after the model discretization is changed. The choice is left to the user's discretion.

For 2D MT, Randie Mackie's model format is used, with the same modification for natural log resistivity.

6 Model covariance?

The model covariance file for 3D MT is rather flexible. It allows the user to specify a smoothing in X and Y directions in each of the vertical layers of the model, and separately to specify vertical smoothing. One can also switch off the smoothing across a boundary, or set is to a specified value. There is also an option to define air and ocean cells in the model covariance file.

The file format is very simple; examples are provided in `examples/3D_MT/` and have the extension of `*.cov`. The file has a 16-line header that is ignored by the code. The header contains a full file format description:

```
+-----------------------------------------------------------------------------+
| This file defines model covariance for a recursive autoregression scheme.   |
| The model space may be divided into distinct areas using integer masks.     |
| Mask 0 is reserved for air; mask 9 is reserved for ocean. Smoothing between |
| air, ocean and the rest of the model is turned off automatically. You can   |
| also define exceptions to override smoothing between any two model areas.   |
| To turn off smoothing set it to zero. This header is 16 lines long.         |
| 1. Grid dimensions excluding air layers (Nx, Ny, NzEarth)                   |
| 2. Smoothing in the X direction (NzEarth real values)                       |
| 3. Smoothing in the Y direction (NzEarth real values)                       |
| 4. Vertical smoothing (1 real value)                                        |
| 5. Number of times the smoothing should be applied (1 integer >= 0)         |
| 6. Number of exceptions (1 integer >= 0)                                    |
| 7. Exceptions in the form e.g. 2 3 0. (to turn off smoothing between 3 & 4) |
| 8. Two integer layer indices and Nx x Ny block of masks, repeated as needed.|
+-----------------------------------------------------------------------------+
```

If no model covariance file is used, the code does something vaguely reasonable by default.

For 2D MT, Weerachai Siripuvaraporn's model covariance is used in the code. No configuration file is currently provided.

7 Known bugs

The implementation of the model covariance had a known bug: applying the model covariance more than once didn't work. More specifically, applying a smoothing that was too strong resulted in unreasonably large variations in the *smoothed* model, and the program crashed. **UPDATE**: Fixed in rev. 326 (trunk rev. 319) by rescaling the covariance and NLCG parameters. Expect NLCG to behave differently from this revision on.

Air layers are currently hard coded in the program. This choice was driven by compatibility with Weerachai's model format. This compatibility has since been partly broken by the addition of the LOGE key in the model file header that allows to store the resistivity values in logarithmic format (LOG10 coming nobody knows when). We are aware of the fact that the user may want to specify the air layers externally. We have not tested our choice of the air layers for accuracy. The thickness of the air layers depends on the top Earth layer in the model. The first air layer matches that, the next one gets multiplied by the factor of 3, and so on for 10 layers total. Thus, the total thickness of the air layers is proportional to the thickness of the top Earth layer.

In 2D MT, the origin was set to THE TOP OF THE AIR LAYERS. This was a serious problem since the vertical site location in the data file had to be defined using distance from the top of the air domain. Z coordinate was therefore dependent on the thickness of the air layers. **UPDATE**: Fixed in rev. 328 (trunk rev. 327) so that zero Z coordinate in the data file corresponds to the Earth's surface; positive is down; negative is up. If we had topography, Earth's surface would be defined by the highest point in the topography and some site Z coordinates would be positive to reflect the topography.

The boundary conditions are inaccurate for 3D MT. The user is advised to place their model boundaries far from all data locations, or use the nested grid feature provided with the modular system.

Excluding some components of a data type from the data file results in unreasonable misfit computations. Until this is fixed, the inversion should only be used with all components of a particular data type. **UPDATE**: Fixed in rev. 333 (trunk rev. 330).

Some care must be exercised when inverting the apparent resistivities and phases in 3D MT. Originally implemented by Kristina Tietze (GFZ); in her implementation, the user had to supply the natural log of apparent resistivities in the file. Stable rev. 371 (trunk rev. 336) reverts to linear apparent resistivities in the file. We now take apparent resistivities from the file and internally convert them to log10 for inversion; output apparent resistivities (no logs). This is to avoid ambiguity in the name of the data functional. The new implementation has not been used. In all likelihood it is correct, but there might still be bugs hiding somewhere in the code.

8 What if something still doesn't work for me?

Please double check that you have compiled the code; that you are running the code on the same system as the one you used to compile it; that you are running the code correctly; that your file formats are correct. Please refer to this user guide for details.

You could also try recompiling the code in the debugging configuration (e.g., by changing the relevant compiler options in your makefile). The output may be more revealing if the program is compiled with debugging options. You could also try running it in the debugger, or using a different compiler.

If you find a reproducible bug, please let us know. Send us the files that are causing the program to misbehave. The authors will try to reproduce and eliminate the bug. Please note that we make no commitment as to the time frame.

Feature requests will be noted and possibly implemented sooner or later. The users are welcome to work with us to improve specific features of ModEM. Any such improvements will be subject to the COPYRIGHT statement as attached.