Open-source 3D forward modeling of MT/RMT considering diffusion and wave fields in anisotropic media

em3d-MT

User Guide Version 1.0

(Updated April 9, 2025)

Contents

1 Introduction

em3d-MT is an open-source MATLAB-based package for 3D forward modeling of magnetotelluric (MT) and radiomagnetotelluric (RMT) data. It is built on an edge-based finite element (FE) framework and solves the full-wave Maxwell's equations, accounting for both conduction and displacement currents. This enables accurate modeling in the RMT frequency range.

The software supports arbitrary anisotropy in both electrical conductivity and dielectric permittivity, allowing flexible simulation of complex geological scenarios. It utilizes unstructured tetrahedral meshes and a customizable geological modeling workflow to accommodate irregular terrain and heterogeneous media.

To address the high computational cost of 3D modeling, em3d-MT integrates a direct linear solver with a double-layer parallelization strategy. This design supports efficient simulation over multiple frequencies using OpenMP-based multithreading and frequency-level parallelization.

2 Installation

This section provides detailed instructions for installing and configuring the em3d-MT package. The software is developed in MATLAB and requires proper setup of external solvers for efficient computation.

2.1 System Requirements

Operating System: Windows or Linux

MATLAB Version: R2023a (or later)

Memory: The actual memory usage depends on model size and number of frequencies.

2.2 Required Software

MATLAB (R2023a or later): The core environment in which em3d-MT operates.

em3d-MT relies on several external tools and libraries to facilitate 3D geological modeling and mesh generation. The following software is required for proper functionality:

GeoMesh

For 3D geological modeling and mesh generation, GeoMesh is used. This is an open-source tool specifically designed for electromagnetic (EM) modeling, based on the COMSOL Multiphysics and MATLAB interface (Liu et al., 2024). GeoMesh generates unstructured tetrahedral meshes, which are ideal for representing complex subsurface structures in the forward modeling process. You can find the tool at https://gitee.com/sduem/geomesh

Additionally, em3d-MT supports two types of direct solvers for solving the finite element system:

Panua-Pardiso:

An optimized direct solver used for solving large sparse linear systems. It offers high performance and memory efficiency. More information can be found at https://panua.ch/pardiso/

UMFPACK:

An alternative direct solver supported by em3d-MT for solving sparse linear systems, suitable for small models or testing. More information can be found at http://faculty.cse.tamu.edu/davis/suitesparse.html

2.3 Installation

Step 1: Install MATLAB

Install MATLAB (R2023a or later). Parallel Computing Toolbox is available for multi-core support.

Step 2: Set Up Panua-Pardiso Solver

Windows: Refer to Barbara (2025). *Add PARDISO lib to Matlab in Windows and LINUX*, MATLAB Central File Exchange. https://www.mathworks.com/matlabcentral/fileexchange/119053-add-pardiso-lib-to-matlab-in-windows-and-linux, January 13, 2025.

The following steps outline how to configure the Panua-Pardiso solver in the Windows environment:

[1] Register for a Panua account. Then download the panua-pardiso-20240228-win package from the official website: https://panua.ch/pardiso/, and extract the contents.

[2] Open the bin folder in the extracted directory. Press Win + R, type cmd, and drag get_fingerprint.exe into the Command Prompt window. Press Enter to generate your machine fingerprint.

[3] Copy the machine fingerprint into the input box on the Panua Pardiso website. An activation key will be sent to your registered email.

[4] Create a file named panua.lic, paste the activation key into the file, and move it to the user directory (e.g., C:\Users\Username).

[5] Add MinGW Compiler to MATLAB by visiting: https://www.mathworks.com/matlabcentral/fileexchange/52848-matlab-support-for-mingw-w64-c-c-compiler

[6] Download and install Intel OneAPI Base Toolkit including MKL (Math Kernel Library) from: https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html

[7] Compile MEX Files. After updating the specific paths in the runMEM.m

script to match your system, run the script in MATLAB to compile the required .mex files.

The following script is used to compile the Panua-Pardiso interface with MATLAB via MEX. It links the necessary Panua, MATLAB, and Intel MKL libraries using MinGW:

```
%% This one works (pardisoinit):
mex -largeArrayDims -L'PATH\TO\YOUR \Pardiso\panua-pardiso-20230718-win\lib' -
llibpardiso...
   -L'PATH\TO\YOUR\Matalb\Matlab2023a\extern\lib\win64\mingw64' -lmwlapack...
   -L'PATH\TO\YOUR\Matalb\Matlab2023a\extern\lib\win64\mingw64' -lmwblas...
   -L'PATH\TO\YOUR\Intel oneAPI\compiler\2023.0.0\windows\compiler\lib\intel64' -lm...
   -output pardisoinit common.cpp matlabmatrix.cpp sparsematrix.cpp pardisoinfo.cpp
pardisoinit.cpp

%% This one works (pardisoreorder):
mex -largeArrayDims -L'PATH\TO\YOUR \Pardiso\panua-pardiso-20230718-win\lib' -
llibpardiso...
   -L'PATH\TO\YOUR\Matalb\Matlab2023a\extern\lib\win64\mingw64' -lmwlapack...
   -L'PATH\TO\YOUR\Matalb\Matlab2023a\extern\lib\win64\mingw64' -lmwblas...
   -L'PATH\TO\YOUR\Intel oneAPI\compiler\2023.0.0\windows\compiler\lib\intel64' -lm...
    -output pardisoreorder common.cpp matlabmatrix.cpp sparsematrix.cpp
pardisoinfo.cpp pardisoreorder.cpp
%% This one works (pardisofactor):
mex -largeArrayDims -L'PATH\TO\YOUR \Pardiso\panua-pardiso-20230718-win\lib' -
llibpardiso...
   -L'PATH\TO\YOUR\Matalb\Matlab2023a\extern\lib\win64\mingw64' -lmwlapack...
   -L'PATH\TO\YOUR\Matalb\Matlab2023a\extern\lib\win64\mingw64' -lmwblas...
   -L'PATH\TO\YOUR\Intel oneAPI\compiler\2023.0.0\windows\compiler\lib\intel64' -lm...
     -output pardisofactor common.cpp matlabmatrix.cpp sparsematrix.cpp
pardisoinfo.cpp pardisofactor.cpp

%% This one works (pardisofactor):
mex -largeArrayDims -L'PATH\TO\YOUR \Pardiso\panua-pardiso-20230718-win\lib' -
llibpardiso...
   -L'PATH\TO\YOUR\Matalb\Matlab2023a\extern\lib\win64\mingw64' -lmwlapack...
   -L'PATH\TO\YOUR\Matalb\Matlab2023a\extern\lib\win64\mingw64' -lmwblas...
   -L'PATH\TO\YOUR\Intel oneAPI\compiler\2023.0.0\windows\compiler\lib\intel64' -lm...
-output pardisofree common.cpp matlabmatrix.cpp sparsematrix.cpp pardisoinfo.cpp
pardisofree.cpp
```

Ubuntu： Refer to the official README provided in the Panua-Pardiso package.

INSTALLATION

This interface for Panua-Pardiso was created using the Matlab external ('mex') interface. For more information on mex, consult
the MathWorks website at https://www.mathworks.com/help/matlab/call-mex-file-functions.html. In order to use this interface in
Matlab, you will first have to configure mex on your system. This can be done by typing in the Matlab command line

```
>> mex -setup
```

The easiest way to install the interface is to use the provided Makefile and type in the command line

```
$ make all
```

To remove all mex executables type in the command line

```
$ make clean
```

The provided interface is currently compatible with Linux users. The Makefile specifies four variables
that you might need to adapt according to your system setup. These are

1. MEXSUFFIX: the suffix appended to the MEX files on your system. For more information type in the Matlab command line
```
>> ext = mexext
```
or visit https://mathworks.com/help/matlab/ref/mexext.html
2. MEX: the mex executable located somewhere within your MATLAB installation directory.
3. CXX: your C++ compiler (executable).
IMPORTANT: The linked C++ compiler must be compatible with your MATLAB installation. MathWorks keeps a detailed list of MATLAB
software package versions and the compilers that are compatible with them here: https://mathworks.com/support/requirements/supported-compilers-linux.html.
4. PARDISOHOME: the directory where the Pardiso shared library is located.
SUGGESTED: your home directory "${HOME}".

If you are not using the provided Makefile, you can compile file individually from the command line. The example below
creates the mex executable pardisoinit.mexa64
$ mex -cxx CXX=g++ CC=g++ LD=g++ -L\~\ -lpardiso -lmwlapack -lmwblas -lgfortran -lpthread -lm -output pardisoinit common.cpp
matlabmatrix.cpp sparsematrix.cpp pardisoinfo.cpp pardisoinit.cpp
Depending on your system setup you may need to include other flags, such as -largeArrayDims.

\------------
CONTENTS
\------------

```
 ├─── common.h, .cpp             # Commonly used mex check functions.
 ├─── matlabmatrix.h, .cpp       # Functions to ensure the compatibility of Pardiso with
Matlab matrices.
 ├─── pardisoinfo.h, .cpp        # Available functionalities of the Pardiso solver.
 ├─── pardisoinit.cpp            # Initialize the Pardiso data structures. Specify type of
matrix, and solver. Produces executable.
    ├─── pardisoinit.mexa64
 ├─── pardisofree.cpp            # Release memory associated with all internal Pardiso
structures. Produces executable.
    ├─── pardisofree.mexa64
 ├─── pardisoreorder.cpp         # Pardiso reordering (symbolic factorization) of sparse
input matrix. Produces executable.
    ├─── pardisoreorder.mexa64
 ├─── pardisofactor.cpp          # Numeric factorization of a sparse matrix. Produces
executable.
    ├─── pardisofactor.mexa64
 ├─── pardisosolve.cpp           # Pardiso solution for a system of equations. Produces
executable.
    ├─── pardisosolve.mexa64
 ├─── example_complex.m          # Using Pardiso on a sparse, symmetric, and complex
matrix.
 ├─── example_hermitian.m        # Using Pardiso on a sparse complex Hermitian
positive definite matrix.
 ├─── example_symmetric.m        # Using Pardiso on a sparse real and symmetric
matrix.
 ├─── example_unsymmetrix.m      # Using Pardiso on a sparse real and unsymmetric
matrix.
```

\------------
NOTES
\------------

1. Discrepancy in OMP version used.
A common error with mex files is the version of OMP that is statically linked to the program. This error would appear

after executing the mex file and would read:

```
>> OMP: Error #15: Initializing libiomp5.a, but found libiomp5.so already initialized.
   OMP: Hint This means that multiple copies of the OpenMP runtime have been linked
into the program.
   That is dangerous, since it can degrade performance or cause incorrect results.
```

To work around this issue set the environment variable KMP_DUPLICATE_LIB_OK to TRUE, in order to allow the program
to continue its execution. In the Matlab command line this is achieved by typing

```
>> setenv("KMP_DUPLICATE_LIB_OK","TRUE")
```

2. Controlling the number of available cores.
The safest way to control the number of cores is by setting the environment variable in the command line, e.g.,

```
$ export OMP_NUM_THREADS=16
```

and then executing Matlab from the same terminal session/window. To check that the number of cores is set
correctly, either use

```
>> verbose = true;
```

in your code, or print out the value of (after Pardiso has been initialized)

```
>> fprintf('Number of cores %d',info.iparm(3));
```

It always recommended to control the parallel execution of the solver by explicitly setting OMP_NUM_THREADS. If fewer
processors are available than specified, the execution may slow down instead of speeding up.
Note that the number of threads that can be utilized by Pardiso is bounded by the maximum number of threads
allowed by your Panua licence. For more information visit http://www.panua.ch/pardiso.

MISC

- The mex interface was first developed by Peter Carbonetto, UBC Vancouver, October 2009. Adapted and modified
by Dimosthenis Pasadakis, Panua-Technologies, January 2024.

3 Source Code Structure

3.1 File Structure

- **bin/**

Contains core source codes:
  - RMT1D/: 1D forward modeling modules
  - RMT3D/: 3D forward modeling modules

  - Solver/: Interfaces to direct solvers

- **examples/**

Contains subdirectories with synthetic examples corresponding to those in the manuscript.

3.2 Computational Workflow

The computational workflow for the em3d-MT model is organized into a series of steps that incorporate mesh processing, material property assignment, matrix assembly, boundary condition application, and the solution of the governing equations. Below is an overview of the computational workflow

Pseudo-code for the em3d-MT Computational Workflow.

```
1:  Input: mesh, P, f, rec, OMP; % mesh, material parameters, frequency, station coordinates, number of threads
2:  mesh = get_mesh_Connect(mesh); % Get grid connection information
3:  Bnd   = get_boundary(mesh); % Get boundary information
4:  [mesh.abcd, mesh.vol] = get_abcd_volume(mesh); % Get the undetermined coefficients and volume
5:  Q    = GetInterpMatrix(rec, mesh); % Get the interpolation matrix of the electromagnetic field
6:  K    = get_K(mesh, P.mur); % Assembled stiffness matrix
7:  M1 = get_M(mesh, P.sig); % Assemble the diffusion field mass matrix
8:  M2 = get_M(mesh, P.eps); % Assemble the wave field mass matrix
9:  boundModel  = get_Boundary1DModel(P, mesh); % The 1D background model was extracted
10: for ifreq = 1:nfeq
11:     $A = K-\omega^2\mu_0 M2-i\omega\mu_0 M1$; % Assemble the coefficient matrix
12:     beField = get_BoundaryFieldAniEpsMu (f, boundModel, Bnd); % Calculation of boundary electric fields
13:     [A, b] = set_Boundary(beField, Bnd, A, b); % Set the boundary condition
14: end
15: parfor ifreq = 1:nfeq
16:     Solve $Ax_{1,2} = b_{1,2}$; % Solve the linear systems of equations of the two polarization modes
17: end
18: data = get_data($x_1$, $x_2$, Q, f); % get MT responses
```

3.3 Main Variables List

In this section, key variables used in the model are outlined, including their descriptions, sizes, and types. These variables are essential for the calculation of electromagnetic field responses, material properties, and the setup of the FE system. The data is organized in different structures (e.g., mesh, P, FE) or table (data) to efficiently store and manage the computational data across multiple frequencies, mesh elements, and material domains.

Variable Name	**Description**	**Size & Type**
mesh.name	Name of the mesh file	char array
mesh.node2coord	Coordinates of all nodes	nnode × 3, double

mesh.elem2node	Node indices for element	nelem × 4, int32
mesh.entity	Entity indices for element	nelem × 1, int32
mesh.elem2edge	Edge indices for element	nelem × 6, double
mesh.edge2node	Node pairs for each edge	nedge × 2, double
mesh.nelem	Total number of elements	1 × 1, double
mesh.nedge	Total number of edges	1 × 1, double
mesh.elemCen	Centroid coordinates of each element	nelem × 3, double
mesh.edgeLength	Length of each edge	nedge × 1, double
mesh.edgeCen	Center coordinates of each edge	nedge × 3, double
mesh.edgeUnit	Unit vector of each edge	nedge × 3, double
mesh.elemEdgeLength	Edge lengths of each element	nelem × 6, double
mesh.sign	Sign of each edge in the element	nelem × 6, double
mesh.abcd	Coefficients for basis functions	nelem × 16, double
mesh.vol	Volume of each element	nelem × 1, double
P.sig_air	Conductivity of air (S/m)	1 × 1, double
P.sigp	Inverse of background conductivity	1 × 1, double
P.sig_anoID	Material IDs of anomalies	nmat (number of material domains) × 1, int32
P.sigax	Conductivity along the X-axis for each anomalous body	nmat × 1, double
P.sigay	Conductivity along the Y-axis for each anomalous body	nmat × 1, double
P.sigaz	Conductivity along the Z-axis for each anomalous body	nmat × 1, double
P.aS	strike angle	nmat × 1, double
P.aD	dip angle	nmat × 1, double
P.aL	slant angle	nmat × 1, double
P.AniAngle	Euler rotation angle	nelem × 3, double
P.sig	Conductivity	nelem × 1; nelem × 3; nelem × 9, double
P.mu0	Magnetic permeability of free space	1 × 1, double
P.mur	Relative permeability of materials	nelem × 1, double
P.mur	Permeability of materials	nelem × 1, double
P.epsr_air	Relative permittivity of air	1 × 1, double
P.epsr	Relative permittivity of materials	nelem × 1, double
P.eps	Absolute permittivity of materials	nelem × 1, double
freq	Frequencies used in the simulation	nfreq × 1, double
omega	Angular frequencies	nfreq × 1, double
nreq	Number of frequencies	1 × 1, double
rec	Coordinates of the measurement points (receiver locations)	nrec× 3, Double
nrec	Number of measurement points	1 × 1, double
K	Stiffness matrix from FE assembly	nedge × nedge, sparse matrix
M1	Mass matrix for the diffusion field	nedge × nedge, sparse matrix
M2	Mass matrix for the wave field	nedge × nedge, sparse matrix
FE.A	Coefficient matrix of the linear system at each frequency	nedge × nedge, complex sparse matrix
FE.rhs	Right-hand side of the linear system at each frequency	2 × 1, struct with Ex-, Ey-polarization mode
FE.rhs.TE	Right-hand side vector for the Ex-polarization mode	nedge × 1, complex double
FE.rhs.TM	Right-hand side vector for the Ey-polarization mode	nedge × 1, complex double
x1	Solution of the linear system for the Ex-polarization mode	nedge × 1, complex double

x2	Solution of the linear system for the Ey-polarization mode	nedge × 1, complex double
data	MT response data	(nrec ×nfreq) × 26, table

3.4 Function List

This section provides an overview of the functions used in the em3d-MT workflow. These functions are categorized by their roles in mesh processing, matrix assembly, solving, and post-processing. Each function's description highlights its specific contribution to the overall electromagnetic field simulation, offering users a clear understanding of the code's modular structure.

Function Name	**Module**	**Description**
em3d_MT.m	Main script	The main script performing mesh loading, preprocessing, FE matrix assembly, boundary setting, solving, and post-processing
Input.m	Input script	The script responsible for loading input data, initializing parameters, and setting up material properties for the model
load_comsol	Input	Loads mesh data from a *.mat* file, returning node2coord (node coordinates), elem2node (element-node connections), and entity (entity IDs)
Load_mesh	Input	Loads mesh data from an ASCII *.dat* file.
Set_material_3Ani	Material Setup	Assigns anisotropic material properties (conductivity, permittivity, permeability) to each mesh element based on the domain IDs and rotation angles
get_sig_ani	Material Setup	Applies 3D rotations to adjust the components of the anisotropic tensors (conductivity, permittivity, etc.) based on specified rotation angles (dip, dip angle, azimuth)
get_mesh_Connect	Mesh processing	Retrieves mesh connectivity information, processing the mesh data for later steps
get_boundary	Mesh processing	Extracts boundary edge information for use in boundary condition setting
get_abcd_volume	Mesh processing	Retrieves the volume and coefficient data for the finite element model
GetInterpMatrix	Field interpolation	Computes the interpolation matrix for the electromagnetic field at measurement points
get_K_v2	FE assembly	Assembles the stiffness matrix K for the finite element calculation
get_M_v2	Finite Element Assembly	Assembles the mass matrix M, used for solving the diffusion and wave fields in the finite element method
get_Boundary1DModel	Boundary Condition	Extracts a 1D background model to generate physical parameters associated with the boundary

get_BoundaryFieldAniEpsMu	Boundary Condition	Computes boundary fields considering anisotropic permittivity and permeability
set_Boundary	Boundary Condition	Sets boundary conditions, returning the coefficient matrices and right-hand-side terms for the finite element equations
pardiso_MT	Solvers	Solves the finite element equation using the Pardiso solver (for TE and TM modes)
umfpack_MT	Solvers	Solves the finite element equation using the UMFPACK solver (for TE and TM modes)
qmr (built-in)	Solvers	Solves the linear system using the QMR (Quasi-Minimal Residual) algorithm
Bicgstab (built-in)	Solvers	Solves the linear system using the BiCGStab (Biconjugate Gradient Stabilized) method
ilu (built-in)	Solvers	Computes incomplete LU factorization for preconditioning
get_data_MT	Post-processing	Computes and retrieves the MT (Magnetotelluric) response data for output analysis
save_mesh_mat2dat	output	Converts and saves mesh data from MATLAB .mat format to ASCII
save	output	store variables
Create_VTK	output	Export the resistivity model as a VTK file

4 Run

Place the file pardiso_MT.m into the directory:

In Window System

PATH\TO\YOUR\em3d_MT\bin\Solver\Pardiso\panua-pardiso-20230718-win\lib

In Ubuntu System

PATH/TO/YOUR/em3d_MT/bin/Solver/Pardiso/panua-pardiso-20230718-win/lib

This ensures that the MATLAB solver script can correctly locate the compiled PARDISO library during runtime

4.1 Run In Ubuntu System

Execute the following command in the terminal:

[1] Ensure that the necessary library paths for the Pardiso solver are set correctly.

```
export LD_LIBRARY_PATH=PATH/TO/YOUR/pardiso/panua-pardiso-20230908-
linx/:$LD_LIBRARY_PATH
```

[2] launch MATLAB by running:

```
PATH/TO/YOUR/MATLAB/R2023b/bin/matlab
```

In MATLAB

[3] Once MATLAB is open, navigate to the directory containing the example files:

```
cd PATH/TO/YOUR/em3d_MT/example1
```

[4] Open Input.m and configure the input parameters as required for your simulation.

[5] Modify the run.mlx script:

This script handles the execution of the forward modeling process. You must modify the paths for the solver libraries as shown below to ensure the necessary files are loaded correctly.

```
addpath(genpath(' PATH/TO/YOUR/em3d_MT/bin')); % add em3d-MT paths
addpath(' PATH/TO/YOUR/em3d_MT/bin/Solver/Pardiso/panua-pardiso-20230718-
win/lib'); % Pardiso
```

[6] Execute the script to start the simulation

4.2 Run In Window System

Launch MATLAB by clicking on matlab.exe. In MATLAB, execute the steps similar to how they are done in the Ubuntu system, with the necessary path configurations and input parameter settings.

4.3 Input

The *Input.m* file is used to configure key physical parameters, model structure, and frequency settings before simulation. The modeling and mesh generation are based on the GeoMesh toolbox https://gitee.com/sduem/geomesh, and the mesh file is typically generated and exported in .mat or .dat format using this tool.

An example of a typical Input.m file is as follows:

```
%%  SET PARAMETER
addpath(genpath('D:\em3d_MT'));
mesh.name  = 'mesh_layered'; % Contains grid information for the model, output by
GeoMesh
[mesh.node2coord,mesh.elem2node,mesh.entity] = load_comsol(mesh.name);
%% Set the conductivity
P.airID      = [4];              % Domain ID of air
P.sig_air    = 1e-8;              % Conductivity of air S/m
P.sigp       = 1./[2000];        % Conductivity of background (residual domain)
P.sig_anoID = [2 1];              % Domain ID of the abnormal body
P.sigax      = 1./[20000 20000]; % Conductivity of the abnormal body
P.sigay      = 1./[20000 20000];
P.sigaz      = 1./[20000 20000];
P.sig =
Set_material_3Ani(mesh.entity,P.sig_anoID,P.airID,P.sigax,P.sigay,P.sigaz,P.sig_air,P.sigp
) ;
% Set the Euler rotation Angle of the abnormal body
P.aS   = [0 0];
P.aD   = [45 45];
P.aL   = [0 0];
P.AniAngle = Set_material_3Ani(mesh.entity,P.sig_anoID,P.airID,P.aS,P.aD,P.aL,0,0) ;
P.sig  =
get_sig_ani(P.sig(:,1),P.sig(:,2),P.sig(:,3),P.AniAngle(:,1),P.AniAngle(:,2),P.AniAngle(:,3));
%% Set magnetic permitivity
P.mu0 = 4*pi*1e-7 ;  % V·s/ (A·m)
P.mu_anoID   = [];  % Domain ID of the abnormal body
P.mura       = [];  % magnetic permitivity of the abnormal body
P.mur = Set_material_3Ani(mesh.entity,P.mu_anoID,P.airID,P.mura,P.mura,P.mura,1,1) ;
P.mur = P.mur(:,1);
P.mu  = P.mur.*P.mu0 ;
%% Set dielectric permittivity
P.epsr_air   = 1;
P.epsrp      = 8;       % dielectric permittivity of background (residual domain)
P.eps_anoID  = [2 1];   % Domain ID of the abnormal body
P.epsax      = [8 8];   % dielectric permittivity of the abnormal body
P.epsay      = [5 5];
P.epsaz      = [6 6];
P.epsr =
Set_material_3Ani(mesh.entity,P.eps_anoID,P.airID,P.epsax,P.epsay,P.epsaz,P.epsr_air,P
.epsrp) ;
P.epsr =
get_sig_ani(P.epsr(:,1),P.epsr(:,2),P.epsr(:,3),P.AniAngle(:,1),P.AniAngle(:,2),P.AniAngle(:,
3));
P.eps0 = 8.854187817620389*10^-12 ;
P.eps  = P.epsr *P.eps0 ;
%% Set frequency
freq  = logspace(4,6,11);
omega = 2*pi*freq;
nfreq = size(freq,2);
```

4.4 Output

This program supports customized output and visualization:

(1) Save simulation results

Use the *save* function to store result variables in a .mat file. For example:

```
save FWD data rec freq
```

Use the *writetable* function to store MT responds result variables in a .csv file. For example:

```
writetable(data, 'data.csv');
```

(2) Export model to VTK format

Use the *Create_VTK* function to export the resistivity model as a VTK file for visualization in tools like ParaView. Example:

```
Create_VTK(mesh.node2coord(:,1)./1000, ...
        mesh.node2coord(:,2)./1000, ...
        mesh.node2coord(:,3)./1000, ...
        (1./P.sig(:,1)), ...
        'model.vtk', '', ...
        mesh.elem2node - 1);
```

Note: The coordinates are converted to kilometers for geophysical-scale 3D visualization.

5 Notes

5.1 Solver Selection

The computational performance reported in the manuscript is based on the Panua-Pardiso solver.

Recommended Solver

Solver Name	Status	Notes
Panua-Pardiso	Preferred recommended	High efficiency and robustness for large-scale, alls-frequency simulations.
UMFPACK	Moderate recommended	Suitable only for small-scale test cases.
Iterative (iluQMR)	Not recommended	May fail to converge below 100 Hz or above 10 kHz.
Iterative (iluBiCGStab)	Not recommended	Similar instability as iluQMR in wide frequency ranges.

How to Set the Solver

In the script, the solver can be specified using the following command:

```
solver = 'Paradiso'; % Options: 'Paradiso', 'umfpack', 'iluQMR', 'iluBicgstab'
```

- Panua-Pardiso is strongly recommended for production runs. If you encounter licensing or compatibility issues, please consult its official documentation or installation guide.
- UMFPACK is available by default in MATLAB and can be used for basic testing or compatibility checks.
- Iterative solvers (iluQMR, iluBiCGStab) may fail to converge below 100 Hz or above

10 kHz.

5.2 Coordinate System

In the code, the Z-axis is defined such that the minimum value of the Z-axis corresponds to the top boundary, and the maximum value corresponds to the bottom boundary. Therefore, for the input mesh coordinates, it is required that Z-values increase downward — i.e., Z is negative upwards and positive downwards.